Managed identities for Azure resources documentation

Learn how to use managed identities in Microsoft Entra ID.

About managed identities

OVERVIEW

What is managed identities for Azure resources?

Configure managed identities on Azure virtual machines

HOW-TO GUIDE

Portal

CLI

PowerShell

Azure Resource Manager Template

REST

Use managed identities on VMs

HOW-TO GUIDE

Acquire an access token

Sign in to PowerShell and CLI

Use with Azure SDK

Configuring applications

HOW-TO GUIDE

Configure an application to trust a managed identity (preview)

Assign a managed identity access to another Azure resource

HOW-TO GUIDE

Portal

CLI

PowerShell

What are managed identities for Azure resources?

Article • 04/11/2025

A common challenge for developers is the management of secrets, credentials, certificates, and keys used to secure communication between services. Manual handling of secrets and certificates are a known source of security issues and outages. Managed identities eliminate the need for developers to manage these credentials. Applications can use managed identities to obtain Microsoft Entra tokens without having to manage any credentials.

What are managed identities?

At a high level, there are two types of identities: human and machine/non-human identities. Machine / non-human identities consist of device and workload identities. In Microsoft Entra, workload identities are applications, service principals, and managed identities. For more information on workload identities, see workload identities.

A managed identity is an identity that can be assigned to an Azure compute resource (Virtual Machine (VM), Virtual Machine Scale Set (VMSS), Service Fabric Cluster, Azure Kubernetes cluster) or any App hosting platform supported by Azure. Once a managed identity is assigned on the compute resource, it can be authorized, directly or indirectly, to access downstream dependency resources, such as a storage account, SQL database, CosmosDB, and so on. Managed identity replaces secrets such as access keys or passwords. In addition, managed identities can replace certificates or other forms of authentication for service-to-service dependencies.

The following video shows how you can use managed identities:
https://learn-video.azurefd.net/vod/player?show=on-net&ep=using-azure-managed-identities&locale=en-us&embedUrl=%2Fentra%2Fidentity%2Fmanaged-identities-azure-resources%2Foverview

Here are some of the benefits of using managed identities:

- You don't need to manage credentials. Credentials aren’t even accessible to you.
- You can use managed identities to authenticate to any resource that supports Microsoft Entra authentication, including your own applications.
- Managed identities can be used at no extra cost.

Managed identity types

There are two types of managed identities:

- **System-assigned**. Some Azure resources, such as virtual machines allow you to enable a managed identity directly on the resource. When you enable a system-assigned managed identity:
  - A service principal of a special type is created in Microsoft Entra ID for the identity. The service principal is tied to the lifecycle of that Azure resource. When the Azure resource is deleted, Azure automatically deletes the service principal for you.
  - By design, only that Azure resource can use this identity to request tokens from Microsoft Entra ID.
  - You authorize the managed identity to have access to one or more services.
  - The name of the system-assigned service principal is always the same as the name of the Azure resource it's created for. For a deployment slot, the name of its system-assigned identity is `<app-name>/slots/<slot-name>`.

- **User-assigned**. You may also create a managed identity as a standalone Azure resource. You can create a user-assigned managed identity and assign it to one or more Azure Resources. When you enable a user-assigned managed identity:
  - A service principal of a special type is created in Microsoft Entra ID for the identity. The service principal is managed separately from the resources that use it.
  - User-assigned identities can be used by multiple resources.
  - You authorize the managed identity to have access to one or more services.

  User-assigned identities, which are provisioned independently from compute and can be assigned to multiple compute resources, are the recommended managed identity type for Microsoft services.

Resources that support system assigned managed identities allow you to:

- Enable or disable managed identities at the resource level.
- Use role-based access control (RBAC) to grant permissions.
- View the create, read, update, and delete (CRUD) operations in Azure Activity logs.
- View sign in activity in Microsoft Entra ID sign in logs.

If you choose a user assigned managed identity instead:

- You can create, read, update, and delete the identities.
- You can use RBAC role assignments to grant permissions.
- User assigned managed identities can be used on more than one resource.
- CRUD operations are available for review in Azure Activity logs.
- View sign in activity in Microsoft Entra ID sign in logs.

Operations on managed identities can be performed by using an Azure Resource Manager template, the Azure portal, Azure CLI, PowerShell, and REST APIs.

Differences between system-assigned and user-assigned managed identities

Expand table

Property	System-assigned managed identity	User-assigned managed identity
Creation	Created as part of an Azure resource (for example, Azure Virtual Machines or Azure App Service).	Created as a stand-alone Azure resource.
Life cycle	Shared life cycle with the Azure resource that the managed identity is created with. When the parent resource is deleted, the managed identity is deleted as well.	Independent life cycle. Must be explicitly deleted.
Sharing across Azure resources	Can't be shared. It can only be associated with a single Azure resource.	Can be shared. The same user-assigned managed identity can be associated with more than one Azure resource.
Common use cases	Workloads contained within a single Azure resource. Workloads needing independent identities. For example, an application that runs on a single virtual machine.	Workloads that run on multiple resources and can share a single identity. Workloads needing preauthorization to a secure resource, as part of a provisioning flow. Workloads where resources are recycled frequently, but permissions should stay consistent. For example, a workload where multiple virtual machines need to access the same resource.

How can I use managed identities for Azure resources?

You can use managed identities by following the steps below:

1. Create a managed identity in Azure. You can choose between system-assigned managed identity or user-assigned managed identity.
   a. When using a user-assigned managed identity, you assign the managed identity to the "source" Azure Resource, such as a Virtual Machine, Azure Logic App or an Azure Web App.
2. Authorize the managed identity to have access to the "target" service.

3. Use the managed identity to access a resource. In this step, you can use the Azure SDK with the Azure.Identity library. Some "source" resources offer connectors that know how to use Managed identities for the connections. In that case, you use the identity as a feature of that "source" resource.

Which Azure services support the feature?

Managed identities for Azure resources can be used to authenticate to services that support Microsoft Entra authentication. For a list of supported Azure services, see services that support managed identities for Azure resources.

Work with managed identities

Managed identities can be used directly or as a Federated Identity Credential for Microsoft Entra ID applications.

The steps involved in using managed identities are as follows:

1. Create a managed identity in Azure. You can choose between system-assigned managed identity or user-assigned managed identity. When using a user-assigned managed identity, you assign the managed identity to the source Azure Resource, such as a Virtual Machine, Azure Logic App or an Azure Web App.
2. Authorize the managed identity to have access to the target service.
3. Use the managed identity to access a resource. In this step, you can use any of the client libraries. Some source resources offer connectors that know how to use Managed identities for the connections. In that case, you use the identity as a feature of that source resource.

Use managed identity directly

Service code running on your Azure compute resource uses either the Microsoft Authentication Library (MSAL) or Azure.Identity SDK to retrieve a managed identity token from Entra ID backed by the managed identity. This token acquisition doesn't require any secrets and is automatically authenticated based on the environment where the code runs. As long as the managed identity is authorized, the service code can access downstream dependencies that support Entra ID authentication.

For example, you can use an Azure Virtual Machine (VM) as Azure Compute. You can then create a user-assigned managed identity and assign it to the VM. The workload running on the VM interfaces with both Azure.Identity (or MSAL) and Azure Storage client SDKs to access a

storage account. The user-assigned managed identity is authorized to access the storage account.

Use managed identity as a Federated Identity Credential (FIC) on an Entra ID app

Workload Identity Federation enables using a managed identity as a credential, just like certificate or password, on Entra ID Applications. Whenever an Entra ID app is required, this is the recommended way to be credential-free. There's a limit of 20 FICs when using managed identities as FIC on an Entra ID App.

A workload acting in the capacity of Entra ID application can be hosted on any Azure compute which has a managed identity. The workload uses the managed identity to acquire a token to be exchanged for an Entra ID Application token, via workload identity federation. This feature is also referred to as managed identity as FIC (Federated Identity Credentials). For more information, see configure an application to trust a managed identity.

Next steps

- Developer introduction and guidelines
- Use a VM system-assigned managed identity to access Resource Manager
- How to use managed identities for App Service and Azure Functions
- How to use managed identities with Azure Container Instances
- Implementing managed identities for Microsoft Azure Resources
- Use workload identity federation for managed identities to access Microsoft Entra protected resources without managing secrets

Secretless authentication for Azure resources

Article • 05/13/2025

Passwords and security keys were the foundation of digital security for decades, but they can no longer keep up with modern threats. Secretless authentication, aligned with the Zero Trust model, shifts access control and user verification to a credential-free approach. Secretless authentication involves designing secure applications in the cloud without relying on traditional, shared credentials such as passwords, certificates, secrets, or security keys. Secretless authentication provides the following benefits:

- Reduced credential risks: By eliminating passwords, secretless authentication mitigates risks associated with credential theft, phishing attacks, and brute force attacks. This approach uses verifiable identity elements such as biometrics, digital certificates, and hardware tokens.
- Streamlined access control: It enhances security and streamlines the user experience by relying on mechanisms that verify true identity rather than shared secrets. This aligns with the principles of Zero Trust by verifying every access request based on true identity.
- Improved end-user experience: Users benefit from a more seamless and secure authentication process, which reduces the need for password resets and improves overall user satisfaction.

This article explores secretless authentication in Azure, its benefits, and how to implement it across various scenarios, including client applications, Azure service-to-service communication, and external workloads.

Security challenges of passwords and secrets

Passwords and other secrets should be used with caution, and developers must never place them in an unsecure location. Many apps connect to backend database, cache, messaging, and eventing services using usernames, passwords, and access keys. If exposed, these credentials could be used to gain unauthorized access to sensitive information such as a sales catalog that you built for an upcoming campaign, or customer data that must be private.

Embedding passwords in an application itself presents a huge security risk for many reasons, including discovery through a code repository. Many developers externalize such passwords using environment variables so that applications can load them from different environments. However, this only shifts the risk from the code itself to an execution environment. Anyone who gains access to the environment can steal passwords, which in turn, increases your data exfiltration risk.

Security challenges of keys

Using cryptographic keys in Azure application development presents several challenges that can complicate both security and operational efficiency. One major issue is key management complexities, which involve the cumbersome tasks of rotating, distributing, and securely storing keys across various services and environments. This ongoing process requires dedicated resources and can significantly increase operational costs due to the need for regular maintenance and monitoring. Additionally, there are substantial security risks associated with key exposure or misuse, as unauthorized access due to leaked keys can compromise sensitive data. Furthermore, key-based authentication methods often lack the scalability and flexibility needed for dynamic environments, making it difficult to adapt to changing requirements and scale operations effectively. These challenges highlight the importance of adopting more secure and efficient authentication methods, such as managed identities, to mitigate risks and streamline operations.

Client application (user-facing) accesses Microsoft Entra protected resources

Features like multifactor authentication (MFA) are a great way to secure your organization, but users often get frustrated with the extra security layer on top of having to remember their passwords. Passwordless authentication methods are more convenient because the password is removed and replaced with something you have or something you are or know.

Each organization has different needs when it comes to authentication. Microsoft Entra ID and Azure Government integrate the following passwordless authentication options:

- Windows Hello for Business
- Platform Credential for macOS
- Platform single sign-on (PSSO) for macOS with smart card authentication
- Microsoft Authenticator
- Passkeys (FIDO2)
- Certificate-based authentication

To learn more, read Microsoft Entra passwordless sign-in.

Azure service-to-Azure service (within Azure)

When authenticating between Azure resources (service-to-service authentication), the recommended approach is to use Managed identities for Azure resources. When authenticating and accessing resources across tenants, however, you should set up a managed identity as a federated identity credential on an application.

Accesses Microsoft Entra protected resources (same tenant)

For scenarios where you need service-to-service authentication between Azure resources and the resources are in the same tenant, managed identities and the `DefaultAzureCredential` class in the Azure Identity client library are the recommended option.

A managed identity is an identity that can be assigned to an Azure compute resource (for example Azure Virtual Machines, Azure Functions, or Azure Kubernetes) or any Azure service that supports managed identities. Once a managed identity is assigned on the compute resource, it can be authorized to access downstream dependency resources, such as a storage account, SQL database, Cosmos DB, and so on. Managed identities replace secrets such as access keys or passwords and certificates or other forms of authentication for service-to-service dependencies.

For more information, read What are managed identities for Azure resources?.

Implement `DefaultAzureCredential` and managed identities in your application to create passwordless connections to Azure services through Microsoft Entra ID and Role Based Access control (RBAC). `DefaultAzureCredential` supports multiple authentication methods and automatically determines which should be used at runtime. This approach enables your app to use different authentication methods in different environments (local dev vs. production) without implementing environment-specific code.

For more info on using `DefaultAzureCredential` and a managed identity in your application, read Configure secretless connections between Azure apps and services.

Accesses Microsoft Entra protected resources (across tenants)

Managed identities are recommended for scenarios where you need service-to-service authentication between Azure resources. Managed identities are not supported, however, for when accessing resources across tenants (multitenant apps). Previously, you created a multitenant application with a client secret or certificate as credentials to authenticate and access resources in multiple tenants. This leaves you with significant risks around secrets exposure and adds the burden of storing, rotating, and maintaining certificate lifecycles.

Azure workloads can now use managed identities as federated identity credentials to securely access Microsoft Entra protected resources across tenants without relying on secrets or certificates.

To learn more, read Configure an application to trust a managed identity.

Azure service accesses external or non-Microsoft Entra protected resources

For Azure workloads that authenticate and access resources that aren't protected by Microsoft Entra or Azure services that require a password, secret, key, or certificate for access you can't directly use managed identities. In this case, use Azure Key Vault to store the credentials for the target resource. Use a managed identity from your workload to retrieve the credentials from key vault and access the target resource using the credentials.

For more information, read Authenticate to Key Vault in code.

External workload (outside Azure) accesses Microsoft Entra protected resources

When a software workload is running outside of Azure (for example, on-premises datacenter, on a developer's machine, or in another cloud) and needs to access Azure resources, you can't use an Azure managed identity directly. In the past, you would register an application with the Microsoft identity platform and use a client secret or certificate for the external app to sign in. These credentials pose a security risk and have to be stored securely and rotated regularly. You also run the risk of service downtime if the credentials expire.

You use workload identity federation to configure a user-assigned managed identity or app registration in Microsoft Entra ID to trust tokens from an external identity provider (IdP), such as GitHub or Google. Once that trust relationship is created, your external software workload exchanges trusted tokens from the external IdP for access tokens from Microsoft identity platform. Your software workload uses that access token to access the Microsoft Entra protected resources to which the workload has been granted access. You eliminate the maintenance burden of manually managing credentials and eliminates the risk of leaking secrets or having certificates expire.

To learn more, read Workload identity federation.

Next steps

- Managed identity best practice recommendations
- Passwordless connections for Azure services
- Authenticate .NET apps to Azure services using the Azure Identity library

Tutorial: Use a system-assigned managed identity on a VM to access Azure Resource Manager

Article • 02/28/2025

This quickstart shows you how to use a system-assigned managed identity as a virtual machine (VM)'s identity to access the Azure Resource Manager API. Managed identities for Azure resources are automatically managed by Azure and enable you to authenticate to services that support Microsoft Entra authentication without needing to insert credentials into your code.

You'll learn how to:

- Grant your virtual machine (VM) access to a resource group in Azure Resource Manager
- Get an access token using a virtual machine (VM) identity and use it to call Azure Resource Manager

Use a Windows VM system-assigned managed identity to access resource manager

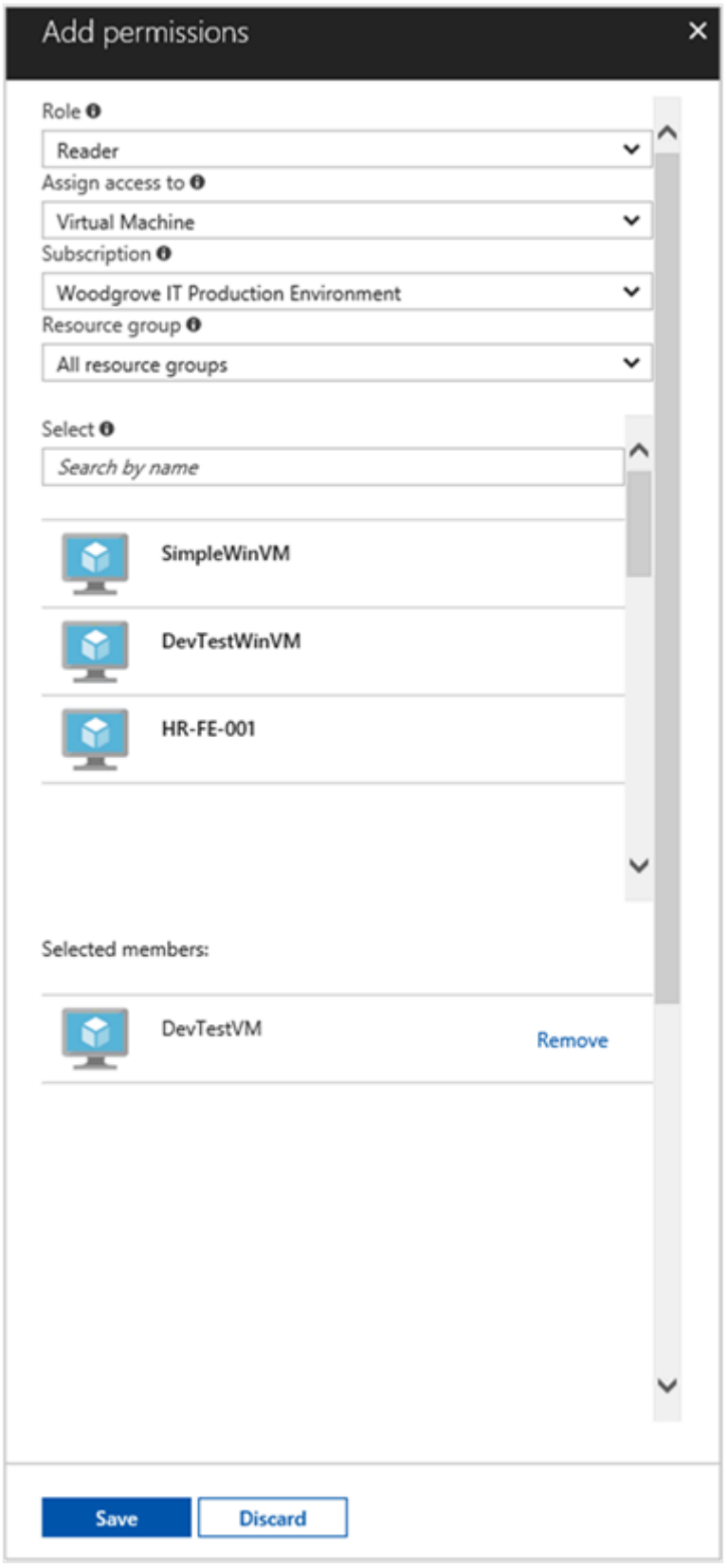
This tutorial explains how to create a system-assigned identity, assign it to a Windows Virtual Machine (VM), and then use that identity to access the Azure Resource Manager API. Managed Service Identities are automatically managed by Azure. They enable authentication to services that support Microsoft Entra authentication, without needing to embed credentials into your code.

You'll learn how to:

- ✓ Grant your VM access to Azure Resource Manager.
- ✓ Get an access token by using the VM's system-assigned managed identity to access Resource Manager.

1. Sign in to the Azure portal with your administrator account.
2. Navigate to the **Resource Groups** tab.
3. Select the **Resource Group** that you want to grant the VM's managed identity access.

4. In the left panel, select **Access control (IAM)**.
5. Select **Add**, then select **Add role assignment**.
6. In the **Role** tab, select **Reader**. This role allows view all resources, but doesn't allow you to make any changes.
7. In the **Members** tab, for the **Assign access to** option, select **Managed identity**, then select **+ Select members**.
8. Ensure the proper subscription is listed in the **Subscription** dropdown. For **Resource Group**, select **All resource groups**.
9. For the **Manage identity** dropdown, select **Virtual Machine**.
10. For **Select**, choose your VM in the dropdown, then select **Save**.



Get an access token

Use the VM's system-assigned managed identity and call the Resource Manager to get an access token.

To complete these steps, you need an SSH client. If you're using Windows, you can use the SSH client in the Windows Subsystem for Linux. If you need assistance configuring

your SSH client's keys, see How to Use SSH keys with Windows on Azure, or How to create and use an SSH public and private key pair for Linux VMs in Azure.

1. In the portal, navigate to your Linux VM and in the **Overview**, select **Connect**.
2. **Connect** to the VM with the SSH client of your choice.
3. In the terminal window, using `curl`, make a request to the local managed identities for Azure resources endpoint to get an access token for Azure Resource Manager. The `curl` request for the access token is below.

Bash

```
curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https://management.azure.com/' -H Metadata:true
```

> **Note**
>
> The value of the `resource` parameter must be an exact match for what is expected by Microsoft Entra ID. In the case of the Resource Manager's resource ID, you must include the trailing slash on the URI.

The response includes the access token you need to access Azure Resource Manager.

Response:

JSON

```
{
  "access_token":"eyJ0eXAiOi...",
  "refresh_token":"",
  "expires_in":"3599",
  "expires_on":"1504130527",
  "not_before":"1504126627",
  "resource":"https://management.azure.com",
  "token_type":"Bearer"
}
```

Use this access token to access Azure Resource Manager; for example, to read the details of the resource group to which you previously granted this VM access. Replace the values of `<SUBSCRIPTION-ID>`, `<RESOURCE-GROUP>`, and `<ACCESS-TOKEN>` with the ones you created earlier.

> **Note**

The URL is case-sensitive, so ensure if you are using the exact case as you used earlier when you named the resource group, and the uppercase “G” in “resourceGroup”.

Bash

```
curl https://management.azure.com/subscriptions/<SUBSCRIPTION-
ID>/resourceGroups/<RESOURCE-GROUP>?api-version=2016-09-01 -H
"Authorization: Bearer <ACCESS-TOKEN>"
```

The response back with the specific resource group information:

JSON

```
{
"id":"/subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-
eeeeee4e4e4e/resourceGroups/DevTest",
"name":"DevTest",
"location":"westus",
"properties":
{
  "provisioningState":"Succeeded"
  }
}
```

Next steps

In this quickstart, you learned how to use a system-assigned managed identity on a VM to access the Azure Resource Manager API. To learn more about Azure Resource Manager, see:

Azure Resource Manager

Create, list or delete a user-assigned managed identity using Azure PowerShell

Feedback

Was this page helpful? Yes No

Provide product feedback

Tutorial: Use a Windows VM/VMSS to access Azure resources

Article • 06/11/2024

Managed identities for Azure resources is a feature of Microsoft Entra ID. Each of the Azure services that support managed identities for Azure resources are subject to their own timeline. Make sure you review the availability status of managed identities for your resource and known issues before you begin.

Prerequisites

- An understanding of managed identities. If you're not familiar with the managed identities for Azure resources feature, see this overview.
- An Azure account, sign up for a free account.
- *Owner* permissions at the appropriate scope (your subscription or resource group) to perform required resource creation and role management steps. If you need assistance with role assignment, see Assign Azure roles to manage access to your Azure subscription resources.
- A Windows virtual machine (VM) that has system assigned managed identities enabled.
  - If you need to create a VM for this tutorial, see Create a virtual machine with system-assigned identity enabled.

Use a Windows VM system-assigned managed identity to access Azure Data Lake Store

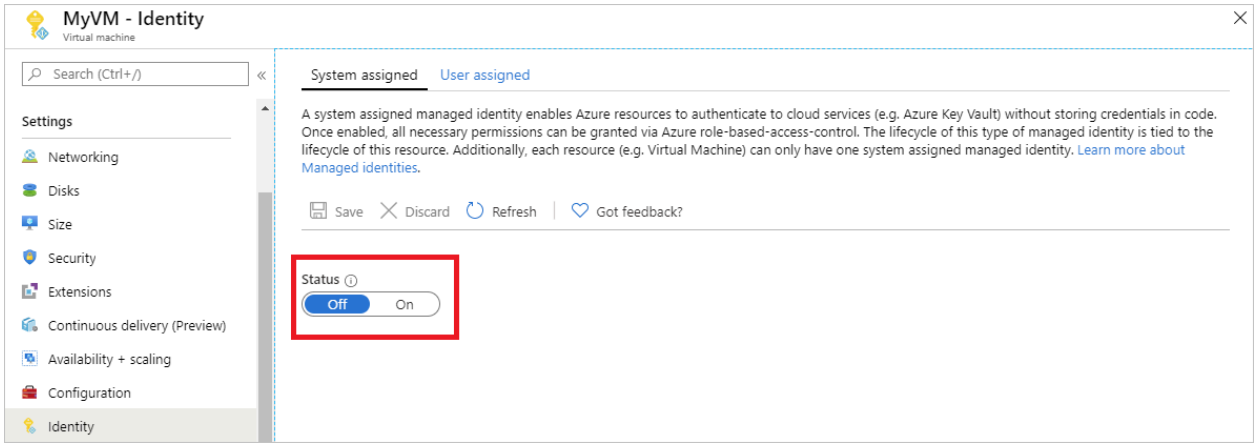
This tutorial shows you how to use a system-assigned managed identity for a Windows virtual machine (VM) to access an Azure Data Lake Store. Managed identities are automatically managed by Azure. They enable your application to authenticate to services that support Microsoft Entra authentication, without needing to insert credentials into your code.

In this article, you'll learn how to:

- ✓ Grant your VM access to an Azure Data Lake Store
- ✓ Get an access token using the VM identity and use it to access an Azure Data Lake Store

Enable

Enabling a system-assigned managed identity is a one-click experience. You can either enable it during the creation of a VM or in the properties of an existing VM.



To enable a system-assigned managed identity on a new VM:

1. Sign in to the Azure portal.
2. Create a virtual machine with system-assigned identity enabled.

Grant access

You can grant your VM access to files and folders in an Azure Data Lake Store. For this step, you can use an existing Data Lake Store or create a new one.

To create a new Data Lake Store using the Azure portal, see Azure Data Lake Store quickstart. There are also quickstarts that use the Azure CLI and Azure PowerShell in the Azure Data Lake Store documentation.

In your Data Lake Store, create a new folder and grant your VM's system-assigned identity permission. The identity needs rights to read, write, and execute files in that folder:

1. In the Azure portal, select **Data Lake Store** in the left-hand navigation.
2. Select the Data Lake Store you want to use for this tutorial.
3. Select **Data Explorer** in the command bar.
4. The root folder of the Data Lake Store is selected. Select **Access** in the command bar.
5. Select **Add**. In the **Select** field, enter the name of your VM, for example **DevTestVM**. Select your VM from the search results, then select **Select**.
6. Select **Select Permissions**, then **Read** and **Execute**. Add to **This folder**, then select **An access permission only**.

7. Select **Ok**, then close the **Access** blade. The permission should be added successfully.
8. Next, create a new folder. Select **New Folder** in the command bar and give the new folder a name. For example, **TestFolder**, then select **Ok**.
9. Select the folder you created, then select **Access** in the command bar.
10. Select **Add**, then in the **Select** field enter the name of your VM and select **Select**.
11. Select **Select Permissions**, then **Read**, **Write** and **Execute**. Add to **This folder**, then add as **An access permission entry and a default permission entry**.
12. Select **Ok**. The permission should be successfully added.

Your VM's system-assigned managed identity can now perform all operations on files in the folder you created. For information on managing access to Data Lake Store, see Access Control in Data Lake Store.

Access data

Azure Data Lake Store natively supports Microsoft Entra authentication, so that it can directly accept access tokens obtained using managed identities for Azure resources. To authenticate to the Data Lake Store filesystem, you send an access token issued by Microsoft Entra ID to your Data Lake Store filesystem endpoint in an Authorization header. The header has the format `Bearer <ACCESS_TOKEN_VALUE>`.

To learn more about Data Lake Store support for Microsoft Entra authentication, see Authentication with Data Lake Store using Microsoft Entra ID.

> **Note**
>
> The Data Lake Store filesystem client SDKs do not yet support managed identities for Azure resources.

In this tutorial, you authenticate to the Data Lake Store filesystem REST API using PowerShell to make REST requests. To use the VM's system-assigned managed identity for authentication, you need to make the requests from the VM.

1. In the portal, navigate to **Virtual Machines**, go to your Windows VM. Then, in the **Overview**, select **Connect**.

2. Enter in your **Username** and **Password** you added when you created the Windows VM.

3. Now that you've created a **Remote Desktop Connection** with the VM, open **PowerShell** in the remote session.

4. Using the PowerShell `Invoke-WebRequest` cmdlet, make a request to the local managed identities for Azure resources endpoint to get an access token for Azure Data Lake Store. The resource identifier for Data Lake Store is `https://datalake.azure.net/`. Data Lake does an exact match on the resource identifier, so the trailing slash is important.

   PowerShell

   ```
   $response = Invoke-WebRequest -Uri
   'http://169.254.169.254/metadata/identity/oauth2/token?api-
   version=2018-02-01&resource=https%3A%2F%2Fdatalake.azure.net%2F' -
   Method GET -Headers @{Metadata="true"}
   ```

   Convert the response from a JSON object to a PowerShell object.

   PowerShell

   ```
   $content = $response.Content | ConvertFrom-Json
   ```

   Extract the access token from the response.

   PowerShell

   ```
   $AccessToken = $content.access_token
   ```

5. Check that everything is configured correctly. Using the PowerShell `Invoke-WebRequest` cmdlet, make a request to your Data Lake Store's REST endpoint to list the folders in the root folder. It's important the string `Bearer` in the Authorization header has a capital "B". You can find the name of your Data Lake Store in the **Overview** section of your Data Lake Store.

   PowerShell

   ```
   Invoke-WebRequest -Uri
   https://<YOUR_ADLS_NAME>.azuredatalakestore.net/webhdfs/v1/?
   op=LISTSTATUS -Headers @{Authorization="Bearer $AccessToken"}
   ```

   A successful response looks like:

   PowerShell

   ```
   StatusCode        : 200
   StatusDescription : OK
   Content           : {"FileStatuses":{"FileStatus":
   ```

```
[{"length":0,"pathSuffix":"TestFolder","type":"DIRECTORY",
"blockSize":0,"accessTime":1507934941392,
"modificationTime":1507944835699,"replication":0,
"permission":"770","ow..."
RawContent        : HTTP/1.1 200 OK
                    Pragma: no-cache
                    x-ms-request-id: b4b31e16-e968-46a1-879a-
3474aa7d4528
                    x-ms-webhdfs-version: 17.04.22.00
                    Status: 0x0
                    X-Content-Type-Options: nosniff
                    Strict-Transport-Security: ma...
Forms             : {}
Headers           : {[Pragma, no-cache], [x-ms-request-id, b4b31e16-
e968-46a1-879a-3474aa7d4528],
                    [x-ms-webhdfs-version, 17.04.22.00], [Status,
0x0]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 556
```

6. Now try uploading a file to your Data Lake Store. First, create a file to upload.

PowerShell

```
echo "Test file." > Test1.txt
```

7. Using the PowerShell `Invoke-WebRequest` cmdlet, make a request to your Data Lake Store's REST endpoint to upload the file to the folder you created earlier. This request takes two steps.
   a. Make a request and get a redirection to where the file should be uploaded.
   b. Upload the file. Remember to set the name of the folder and file appropriately if you used different values than indicated in this tutorial.

PowerShell

```
$HdfsRedirectResponse = Invoke-WebRequest -Uri
https://<YOUR_ADLS_NAME>.azuredatalakestore.net/webhdfs/v1/TestFolder/T
est1.txt?op=CREATE -Method PUT -Headers @{Authorization="Bearer
$AccessToken"} -Infile Test1.txt -MaximumRedirection 0
```

If you inspect the value of `$HdfsRedirectResponse`, it should look like the following response:

PowerShell

```
PS C:\> $HdfsRedirectResponse

StatusCode        : 307
StatusDescription : Temporary Redirect
Content           : {}
RawContent        : HTTP/1.1 307 Temporary Redirect
                    Pragma: no-cache
                    x-ms-request-id: b7ab492f-b514-4483-aada-
4aa0611d12b3
                    ContentLength: 0
                    x-ms-webhdfs-version: 17.04.22.00
                    Status: 0x0
                    X-Content-Type-Options: nosn...
Headers           : {[Pragma, no-cache], [x-ms-request-id, b7ab492f-
b514-4483-aada-4aa0611d12b3],
                    [ContentLength, 0], [x-ms-webhdfs-version,
17.04.22.00]...}
RawContentLength  : 0
```

Complete the upload by sending a request to the redirect endpoint:

PowerShell

```
Invoke-WebRequest -Uri $HdfsRedirectResponse.Headers.Location -Method
PUT -Headers @{Authorization="Bearer $AccessToken"} -Infile Test1.txt -
MaximumRedirection 0
```

A successful response look like:

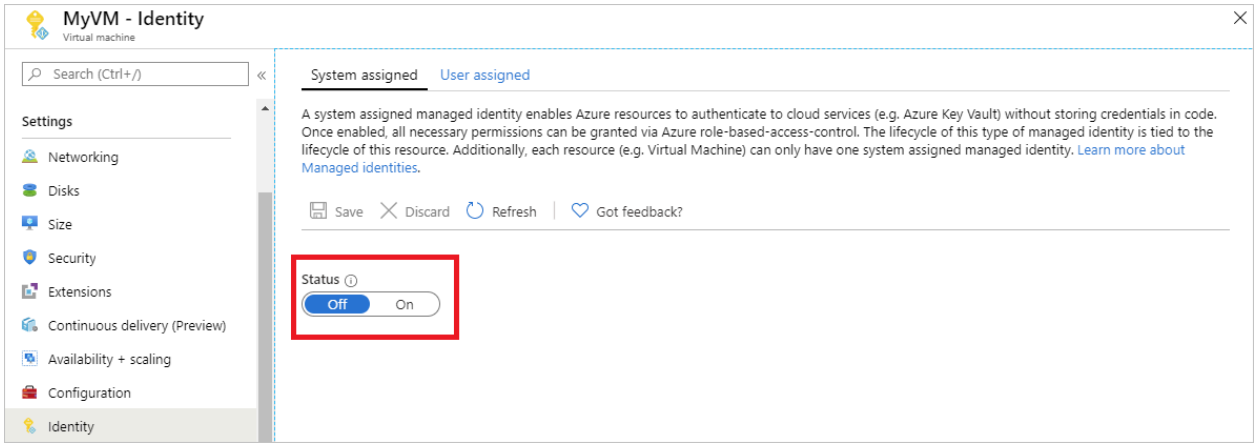
PowerShell

```
StatusCode        : 201
StatusDescription : Created
Content           : {}
RawContent        : HTTP/1.1 201 Created
                    Pragma: no-cache
                    x-ms-request-id: 1e70f36f-ead1-4566-acfa-
d0c3ec1e2307
                    ContentLength: 0
                    x-ms-webhdfs-version: 17.04.22.00
                    Status: 0x0
                    X-Content-Type-Options: nosniff
                    Strict...
Headers           : {[Pragma, no-cache], [x-ms-request-id, 1e70f36f-
ead1-4566-acfa-d0c3ec1e2307],
                    [ContentLength, 0], [x-ms-webhdfs-version,
17.04.22.00]...}
RawContentLength  : 0
```

Finally, you can use other Data Lake Store filesystem APIs to append to and download files, and more.

Disable

To disable the system-assigned identity on your VM, set the status of the system-assigned identity to **Off**.



Learn more

- What are managed identities for Azure resources?
- Quickstart: Use a user-assigned managed identity on a VM to access Azure Resource Manager
- Create, list, or delete a user-assigned managed identity using Azure PowerShell

---

Feedback

Was this page helpful? Yes No

Provide product feedback

Tutorial: Use a Linux VM/VMSS to access Azure resources

Article • 06/11/2024

Managed identities for Azure resources is a feature of Microsoft Entra ID. Each of the Azure services that support managed identities for Azure resources are subject to their own timeline. Make sure you review the availability status of managed identities for your resource and known issues before you begin.

Prerequisites

- An understanding of managed identities. If you're not familiar with the managed identities for Azure resources feature, see this overview.
- An Azure account, sign up for a free account.
- *Owner* permissions at the appropriate scope (your subscription or resource group) to perform required resource creation and role management steps. If you need assistance with role assignment, see Assign Azure roles to manage access to your Azure subscription resources.
- A Linux virtual machine (VM) that has system assigned managed identities enabled.
  - If you need to create a VM for this tutorial, see Create a virtual machine with system-assigned identity enabled.

Use a Linux VM system-assigned managed identity to access Azure Data Lake Store

This tutorial shows you how to use a system-assigned managed identity for a Linux virtual machine (VM) to access Azure Data Lake Store.

You'll learn how to:

- ✓ Grant your VM access to Azure Data Lake Store.
- ✓ Get an access token by using the VM's system-assigned managed identity to access Azure Data Lake Store.

Grant access

This section shows how to grant your VM access to files and folders in Azure Data Lake Store. For this step, you can use an existing Data Lake Store instance or create a new one. To create a Data Lake Store instance by using the Azure portal, follow the Azure Data Lake Store quickstart. There are also quickstarts that use Azure CLI and Azure PowerShell in the Azure Data Lake Store documentation.

In Data Lake Store, create a new folder and grant the Linux VM system-assigned managed identity permission to read, write, and execute files in that folder:

1. In the Azure portal, select **Data Lake Store** in the left pane.
2. Select the Data Lake Store instance that you want to use.
3. Select **Data Explorer** on the command bar.
4. The root folder of the Data Lake Store instance is selected. Select **Access** on the command bar.
5. Select **Add**. In the **Select** box, enter the name of your VM; for example, **DevTestVM**. Select your VM from the search results, then select **Select**.
6. Select **Select Permissions**. Select **Read** and **Execute**, add to **This folder**, add as **An access permission only**, then select **Ok**. The permission should be added successfully.
7. Close the **Access** pane.
8. Create a new folder, then select **New Folder** on the command bar and give the new folder a name; for example, **TestFolder**, then select **Ok**.
9. Select the folder that you created, then select **Access** on the command bar.
10. Select **Add**, then in the **Select** box, enter the name of your VM.
11. Select your VM from the search results, then select **Select**.
12. Select **Select Permissions**, then select **Read**, then **Write** and **Execute**.
13. Select to add to **This folder**, then add as **An access permission entry and a default permission entry**, then select **Ok**. The permission should be added successfully.

Managed identities for Azure resources can now perform all operations on files in the folder that you created. For more information on managing access to Data Lake Store, see Access Control in Data Lake Store.

Get an access token

This section shows how to obtain an access token and call the Data Lake Store file system. Azure Data Lake Store natively supports Microsoft Entra authentication, so it can directly accept access tokens obtained via using managed identities for Azure resources.

To authenticate to the Data Lake Store file system, you send an access token issued by Microsoft Entra ID to your Data Lake Store file system endpoint. The access token is in an authorization header in the format `Bearer \<ACCESS_TOKEN_VALUE\>`. To learn more

about Data Lake Store support for Microsoft Entra authentication, see Authentication with Data Lake Store using Microsoft Entra ID.

Next, you authenticate to the REST API for the Data Lake Store file system by using cURL to make REST requests.

> **Note**
>
> The client SDKs for the Data Lake Store file system do not yet support managed identities for Azure resources.

To complete these steps, you need an SSH client. If you are using Windows, you can use the SSH client in the Windows Subsystem for Linux. If you need assistance configuring your SSH client's keys, see How to use SSH keys with Windows on Azure or How to create and use an SSH public and private key pair for Linux VMs in Azure.

1. In the portal, browse to your Linux VM, then in the **Overview** section, select **Connect**.

2. Connect to the VM by using the SSH client of your choice.

3. In the terminal window, using cURL make a request to the local managed identities Azure for Azure resources endpoint to get an access token for the Data Lake Store file system. The resource identifier for Data Lake Store is `https://datalake.azure.net/`. It's important to include the trailing slash in the resource identifier.

   Bash

   ```
   curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https%3A%2F%2Fdatalake.azure.net%2F' -H Metadata:true
   ```

   A successful response returns the access token that you use to authenticate to Data Lake Store:

   Bash

   ```
   {"access_token":"eyJ0eXAiOiJ...",
    "refresh_token":"",
    "expires_in":"3599",
    "expires_on":"1508119757",
    "not_before":"1508115857",
   ```

```
 "resource":"https://datalake.azure.net/",
 "token_type":"Bearer"}
```

4. By using cURL, make a request to your Data Lake Store file system's REST endpoint to list the folders in the root folder. This is the best way to check that everything is configured correctly. Copy the value of the access token from the previous step. It's important that the string `Bearer` in the Authorization header has a capital "B." You can find the name of your Data Lake Store instance in the **Overview** section of the **Data Lake Store** pane in the Azure portal.

```
Bash

curl https://<YOUR_ADLS_NAME>.azuredatalakestore.net/webhdfs/v1/?
op=LISTSTATUS -H "Authorization: Bearer <ACCESS_TOKEN>"
```

   A successful response looks like this:

```
Bash

{"FileStatuses":{"FileStatus":
[{"length":0,"pathSuffix":"TestFolder","type":"DIRECTORY","blockSize":0
,"accessTime":1507934941392,"modificationTime":1508105430590,"replicati
on":0,"permission":"770","owner":"bd0e76d8-ad45-4fe1-8941-
04a7bf27f071","group":"bd0e76d8-ad45-4fe1-8941-04a7bf27f071"}]}}
```

5. Now upload a file to your Data Lake Store instance. First, create a file to upload.

```
Bash

echo "Test file." > Test1.txt
```

6. By using cURL, make a request to your Data Lake Store file system's REST endpoint to upload the file to the folder that you created earlier. The upload involves a redirect, and cURL follows the redirect automatically.

```
Bash

curl -i -X PUT -L -T Test1.txt -H "Authorization: Bearer
<ACCESS_TOKEN>"
'https://<YOUR_ADLS_NAME>.azuredatalakestore.net/webhdfs/v1/<FOLDER_NAM
E>/Test1.txt?op=CREATE'
```

   A successful response looks like this:

```
Bash
```

```
HTTP/1.1 100 Continue
HTTP/1.1 307 Temporary Redirect
Cache-Control: no-cache, no-cache, no-store, max-age=0
Pragma: no-cache
Expires: -1
Location:
https://mytestadls.azuredatalakestore.net/webhdfs/v1/TestFolder/Test1.t
xt?op=CREATE&write=true
x-ms-request-id: 756f6b24-0cca-47ef-aa12-52c3b45b954c
ContentLength: 0
x-ms-webhdfs-version: 17.04.22.00
Status: 0x0
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=15724800; includeSubDomains
Date: Sun, 15 Oct 2017 22:10:30 GMT
Content-Length: 0

HTTP/1.1 100 Continue

HTTP/1.1 201 Created
Cache-Control: no-cache, no-cache, no-store, max-age=0
Pragma: no-cache
Expires: -1
Location:
https://mytestadls.azuredatalakestore.net/webhdfs/v1/TestFolder/Test1.t
xt?op=CREATE&write=true
x-ms-request-id: af5baa07-3c79-43af-a01a-71d63d53e6c4
ContentLength: 0
x-ms-webhdfs-version: 17.04.22.00
Status: 0x0
X-Content-Type-Options: nosniff
Strict-Transport-Security: max-age=15724800; includeSubDomains
Date: Sun, 15 Oct 2017 22:10:30 GMT
Content-Length: 0
```

Finally, you can now use other APIs for the Data Lake Store file system to append to files, download files, and more.

Learn more

- What are managed identities for Azure resources?
- Quickstart: Use a user-assigned managed identity on a VM to access Azure Resource Manager
- Create, list, or delete a user-assigned managed identity using Azure PowerShell

---

Feedback

Was this page helpful? Yes No

Provide product feedback

Use an Azure managed identity to authenticate to an Azure container registry

Article • 05/04/2025

Use a managed identity for Azure resources to authenticate to an Azure container registry from another Azure resource, without needing to provide or manage registry credentials. For example, set up a user-assigned or system-assigned managed identity on a Linux VM to access container images from your container registry, as easily as you use a public registry. Or, set up an Azure Kubernetes Service cluster to use its managed identity to pull container images from Azure Container Registry for pod deployments.

For this article, you learn more about managed identities and how to:

- ✓ Enable a user-assigned or system-assigned identity on an Azure VM
- ✓ Grant the identity access to an Azure container registry
- ✓ Use the managed identity to access the registry and pull a container image

Azure CLI

To create the Azure resources, this article requires that you run the Azure CLI version 2.0.55 or later. Run `az --version` to find the version. If you need to install or upgrade, see Install Azure CLI.

To set up a container registry and push a container image to it, you must also have Docker installed locally. Docker provides packages that easily configure Docker on any macOS, Windows, or Linux system.

Why use a managed identity?

If you're not familiar with the managed identities for Azure resources feature, see this overview.

After you set up selected Azure resources with a managed identity, give the identity the access you want to another resource, just like any security principal. For example, assign a managed identity a role with pull, push and pull, or other permissions to a private registry in Azure. (For a complete list of registry roles, see Azure Container Registry Entra permissions and roles overview.) You can give an identity access to one or more resources.

Then, use the identity to authenticate to any service that supports Microsoft Entra authentication, without any credentials in your code. Choose how to authenticate using the managed identity, depending on your scenario. To use the identity to access an Azure container registry from a virtual machine, you authenticate with Azure Resource Manager.

Create a container registry

Azure CLI

If you don't already have an Azure container registry, create a registry and push a sample container image to it. For steps, see Quickstart: Create a private container registry using the Azure CLI.

This article assumes you have the `aci-helloworld:v1` container image stored in your registry. The examples use a registry name of *myContainerRegistry*. Replace with your own registry and image names in later steps.

Create a Docker-enabled VM

Azure CLI

Create a Docker-enabled Ubuntu virtual machine. You also need to install the Azure CLI on the virtual machine. If you already have an Azure virtual machine, skip this step to create the virtual machine.

Deploy a default Ubuntu Azure virtual machine with az vm create. The following example creates a VM named *myDockerVM* in an existing resource group named *myResourceGroup*:

Azure CLI

```
az vm create \
    --resource-group myResourceGroup \
    --name myDockerVM \
    --image Ubuntu2204 \
    --admin-username azureuser \
    --generate-ssh-keys
```

It takes a few minutes for the VM to be created. When the command completes, take note of the `publicIpAddress` displayed by the Azure CLI. Use this address to make SSH connections to the VM.

Install Docker on the VM

To run Docker containers on your virtual machine, you need to install Docker. This section provides the steps to install Docker on an Ubuntu VM, ensuring that your VM is ready to pull and run container images from your Azure Container Registry.

After the VM is running, make an SSH connection to the VM. Replace *publicIpAddress* with the public IP address of your VM.

Bash

```
ssh azureuser@publicIpAddress
```

Run the following command to install Docker on the VM:

Bash

```
sudo apt update
sudo apt install docker.io -y
```

After installation, run the following command to verify that Docker is running properly on the VM:

Bash

```
sudo docker run -it mcr.microsoft.com/hello-world
```

Output

```
Hello from Docker!
This message shows that your installation appears to be working correctly.
[...]
```

Azure CLI

Install the Azure CLI

Follow the steps in Install Azure CLI with apt to install the Azure CLI on your Ubuntu virtual machine. For this article, ensure that you install version 2.0.55 or later.

Exit the SSH session.

Example 1: Access with a user-assigned identity

Create an identity

Azure CLI

Create an identity in your subscription using the az identity create command. You can use the same resource group you used previously to create the container registry or virtual machine, or a different one.

Azure CLI

```
az identity create --resource-group myResourceGroup --name myACRId
```

To configure the identity in the following steps, use the az identity show command to store the identity's resource ID and service principal ID in variables.

Azure CLI

```
# Get resource ID of the user-assigned identity
userID=$(az identity show --resource-group myResourceGroup --name myACRId --
query id --output tsv)

# Get service principal ID of the user-assigned identity
spID=$(az identity show --resource-group myResourceGroup --name myACRId --
query principalId --output tsv)
```

Because you need the identity's ID in a later step when you sign in to the CLI from your virtual machine, show the value:

Bash

```
echo $userID
```

The ID is of the form:

Output

```
/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx/resourcegroups/myResourceGroup/providers/Microsoft.ManagedIdenti
ty/userAssignedIdentities/myACRId
```

Configure the VM with the identity

Azure CLI

The following az vm identity assign command configures your Docker VM with the user-assigned identity:

Azure CLI

```
az vm identity assign --resource-group myResourceGroup --name myDockerVM --
identities $userID
```

Grant identity access to the container registry

Azure CLI

Now configure the identity to access your container registry. First use the az acr show command to get the resource ID of the registry:

Azure CLI

```
resourceID=$(az acr show --resource-group myResourceGroup --name
myContainerRegistry --query id --output tsv)
```

Use the az role assignment create command to assign the correct role to the identity. You must assign either `Container Registry Repository Reader` (for ABAC-enabled registries) or `AcrPull` (for non-ABAC registries). This role assignment provides pull permissions to the registry.

To provide both pull and push permissions, assign either the `Container Registry Repository Writer` role for ABAC-enabled registries, or the `AcrPush` role for non-ABAC-enabled registries.

For more information on Microsoft Entra ABAC, see Microsoft Entra-based repository permissions.

Azure CLI

```
az role assignment create --assignee $spID --scope $resourceID \
    --role "Container Registry Repository Reader" # For ABAC-enabled
registries. Otherwise, use AcrPull for non-ABAC registries.
```

Use the identity to access the registry

Azure CLI

SSH into the Docker virtual machine that's configured with the identity. Run the following Azure CLI commands, using the Azure CLI installed on the VM.

First, authenticate to the Azure CLI with az login, using the identity you configured on the VM. For `<userID>`, substitute the ID of the identity you retrieved in a previous step.

Azure CLI

```
az login --identity --username <userID>
```

Then, authenticate to the registry with az acr login. When you use this command, the CLI uses the Active Directory token created when you ran `az login` to seamlessly authenticate your session with the container registry. (Depending on your VM's setup, you might need to run this command and docker commands with `sudo`.)

Azure CLI

```
az acr login --name myContainerRegistry
```

You should see a `Login succeeded` message. You can then run `docker` commands without providing credentials. For example, run docker pull to pull the `aci-helloworld:v1` image, specifying the login server name of your registry. The login server name consists of your container registry name (all lowercase) followed by `.azurecr.io` - for example, `mycontainerregistry.azurecr.io`.

```

docker pull mycontainerregistry.azurecr.io/aci-helloworld:v1
```

Configure the VM with a system-managed identity

A system-assigned managed identity is a feature of Azure that allows your virtual machine to automatically manage its own identity in Azure Active Directory. This section explains how to configure your VM with a system-assigned identity to securely access your Azure Container Registry.

Configure the VM with a system-managed identity

Azure CLI

The following az vm identity assign command configures your Docker VM with a system-assigned identity:

Azure CLI

```
az vm identity assign --resource-group myResourceGroup --name myDockerVM
```

Use the az vm show command to set a variable to the value of `principalId` (the service principal ID) of the VM's identity, to use in later steps.

Azure CLI

```
spID=$(az vm show --resource-group myResourceGroup --name myDockerVM --query
identity.principalId --out tsv)
```

Grant identity access to the container registry

Azure CLI

Now configure the identity to access your container registry. First use the az acr show command to get the resource ID of the registry:

Azure CLI

```
resourceID=$(az acr show --resource-group myResourceGroup --name
myContainerRegistry --query id --output tsv)
```

Use the az role assignment create command to assign the correct role to the identity. You must assign either `Container Registry Repository Reader` (for ABAC-enabled registries) or `AcrPull` (for non-ABAC registries). This role assignment provides pull permissions to the registry.

To provide both pull and push permissions, assign either the `Container Registry Repository Writer` role for ABAC-enabled registries, or the `AcrPush` role for non-ABAC-enabled registries.

For more information on Microsoft Entra ABAC, see Microsoft Entra-based repository permissions.

Azure CLI

```
az role assignment create --assignee $spID --scope $resourceID \
    --role "Container Registry Repository Reader" # For ABAC-enabled
registries. Otherwise, use AcrPull for non-ABAC registries.
```

Use the identity to access the registry

Azure CLI

SSH into the Docker virtual machine that's configured with the identity. Run the following Azure CLI commands, using the Azure CLI installed on the VM.

First, authenticate the Azure CLI with az login, using the system-assigned identity on the VM.

Azure CLI

```
az login --identity
```

Then, authenticate to the registry with az acr login. When you use this command, the CLI uses the Active Directory token created when you ran `az login` to seamlessly authenticate your session with the container registry. (Depending on your VM's setup, you might need to run this command and docker commands with `sudo`.)

Azure CLI

```
az acr login --name myContainerRegistry
```

You should see a `Login succeeded` message. You can then run `docker` commands without providing credentials. For example, run docker pull to pull the `aci-helloworld:v1` image, specifying the login server name of your registry. The login server name consists of your container registry name (all lowercase) followed by `.azurecr.io` - for example, `mycontainerregistry.azurecr.io`.

Bash

```
docker pull mycontainerregistry.azurecr.io/aci-helloworld:v1
```

Next steps

In this article, you learned about using managed identities with Azure Container Registry and how to:

- ✓ Enable a user-assigned or system-assigned identity in an Azure VM
- ✓ Grant the identity access to an Azure container registry
- ✓ Use the managed identity to access the registry and pull a container image

- Learn more about managed identities for Azure resources.
- Learn how to use a system-assigned or user-assigned managed identity with App Service and Azure Container Registry.
- Learn how to deploy a container image from Azure Container Registry using a managed identity.

How to use managed identities to connect to Azure Cosmos DB from an Azure virtual machine

Article • 01/29/2025

In this article, we set up a virtual machine to use managed identities to connect to Azure Cosmos DB. Azure Cosmos DB is a fully managed NoSQL database for modern app development. Managed identities for Azure resources allow your applications to authenticate when accessing services that support Microsoft Entra authentication using an identity managed by Azure.

Prerequisites

- A basic understanding of Managed identities. If you would like to learn more about managed identities for Azure resources before you continue, review the managed identities overview.
- You must have an Azure account with an active subscription. Create an account for free.
- You may need either PowerShell or the CLI.
- Visual Studio Community Edition or some other development environment of your choosing.

Create a resource group

Create a resource group called **mi-test**. We use this resource group for all resources used in this tutorial.

- Create a resource group using the Azure portal
- Create a resource group using the CLI
- Create a resource group using PowerShell

Create an Azure VM with a managed identity

For this tutorial, you need an Azure virtual machine(VM). Create a virtual machine with a system-assigned managed identity enabled called **mi-vm-01**. You may also create a user-assigned managed identity called **mi-ua-01** in the resource group we created

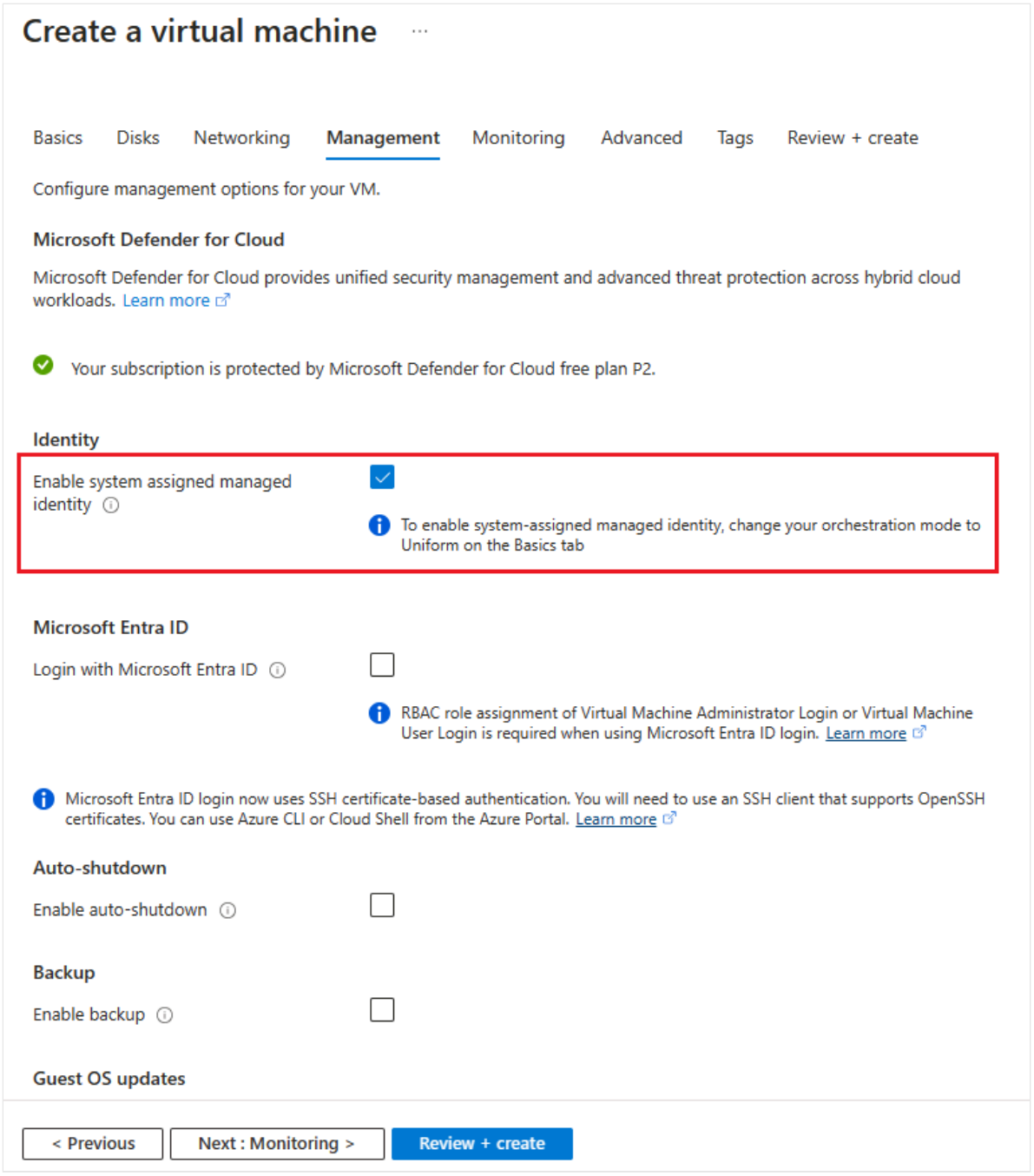
earlier (**mi-test**). If you use a user-assigned managed identity, you can assign it to a VM during creation.

Create a VM with a system-assigned managed identity

To create an Azure VM with the system-assigned managed identity enabled, your account needs the Virtual Machine Contributor role assignment. No other Microsoft Entra role assignments are required.

Portal

- From the **Azure portal** search for **virtual machines**.
- Choose **Create**
- In the Basics tab, provide the required information.
- Choose **Next: Disks >**
- Continue filling out information as needed and in the **Management** tab find the **Identity** section and check the box next to **System assigned managed identity**



For more information, review the Azure virtual machines documentation:

- Linux
- Windows

Create a VM with a user-assigned managed identity

The steps below show you how to create a virtual machine with a user-assigned managed identity configured.

Portal

Today, the Azure portal doesn't support assigning a user-assigned managed identity during the creation of a VM. You should create a virtual machine and then assign a user assigned managed identity to it.

Configure managed identities for Azure resources on a VM using the Azure portal

Create an Azure Cosmos DB account

Now that we have a VM with either a user-assigned managed identity or a system-assigned managed identity we need an Azure Cosmos DB account available where you have administrative rights. If you need to create an Azure Cosmos DB account for this tutorial, the Azure Cosmos DB quickstart provides detailed steps on how to do that.

> **Note**
>
> Managed identities may be used to access any Azure resource that supports Microsoft Entra authentication. This tutorial assumes that your Azure Cosmos DB account will be configured as shown below.

Expand table

Setting	Value	Description
Subscription	Subscription name	Select the Azure subscription that you want to use for this Azure Cosmos DB account.
Resource Group	Resource group name	Select **mi-test**, or select **Create new**, then enter a unique name for the new resource group.
Account Name	A unique name	Enter a name to identify your Azure Cosmos DB account. Because *documents.azure.com* is appended to the name that you provide to create your URI, use a unique name. The name can only contain lowercase letters, numbers, and the hyphen (-) character. It must be between 3-44 characters in length.
API	The type of account to create	Select **Azure Cosmos DB for NoSQL** to create a document database and query by using SQL syntax. Learn more about the SQL API.
Location	The region closest to your	Select a geographic location to host your Azure Cosmos DB account. Use the location that is closest to your users to give

Setting	Value	Description
	users	them the fastest access to the data.

> **Note**
>
> If you are testing you may want to apply Azure Cosmos DB free tier discount. With the Azure Cosmos DB free tier, you will get the first 1000 RU/s and 25 GB of storage for free in an account. Learn more about free tier. Keep in mind that for the purpose of this tutorial this choice makes no difference.

Grant access

At this point, we should have both a virtual machine configured with a managed identity and an Azure Cosmos DB account. Before we continue, we need to grant the managed identity a couple of different roles.

- First grant access to the Azure Cosmos DB management plane using Azure RBAC. The managed identity needs to have the DocumentDB Account Contributor role assigned to create Databases and containers.
- You also need to grant the managed identity a contributor role using Azure Cosmos DB RBAC. You can see specific steps below.

> **Note**
>
> We will use the **Cosmos DB Built-in Data contributor** role. To grant access, you need to associate the role definition with the identity. In our case, the managed identity associated with our virtual machine.

Portal

At this time there is no role assignment option available in the Azure portal

Access data

Getting access to Azure Cosmos DB using managed identities may be achieved using the Azure.identity library to enable authentication in your application. You can call ManagedIdentityCredential directly or use DefaultAzureCredential.

The ManagedIdentityCredential class attempts to authentication using a managed identity assigned to the deployment environment. The DefaultAzureCredential class goes through different authentication options in order. The second authentication option that DefaultAzureCredential attempts is Managed identities.

In the example shown below, you create a database, a container, an item in the container, and read back the newly created item using the virtual machine's system assigned managed identity. If you want to use a user-assigned managed identity, you need to specify the user-assigned managed identity by specifying the managed identity's client ID.

C#

```
string userAssignedClientId = "<your managed identity client Id>";
var tokenCredential = new DefaultAzureCredential(new
DefaultAzureCredentialOptions { ManagedIdentityClientId =
userAssignedClientId });
```

To use the sample below, you need to have the following NuGet packages:

- Azure.Identity
- Microsoft.Azure.Cosmos
- Microsoft.Azure.Management.CosmosDB

In addition to the NuGet packages above, you also need to enable **Include prerelease** and then add **Azure.ResourceManager.CosmosDB**.

C#

```
using Azure.Identity;
using Azure.ResourceManager.CosmosDB;
using Azure.ResourceManager.CosmosDB.Models;
using Microsoft.Azure.Cosmos;
using System;
using System.Threading.Tasks;

namespace MITest
{
    class Program
    {
        static async Task Main(string[] args)
        {
            // Replace the placeholders with your own values
            var subscriptionId = "Your subscription ID";
            var resourceGroupName = "You resource group";
            var accountName = "Cosmos DB Account name";
            var databaseName = "mi-test";
            var containerName = "container01";
```

```
            // Authenticate to Azure using Managed Identity (system-assigned
or user-assigned)
            var tokenCredential = new DefaultAzureCredential();

            // Create the Cosmos DB management client using the subscription
ID and token credential
            var managementClient = new
CosmosDBManagementClient(tokenCredential)
            {
                SubscriptionId = subscriptionId
            };

            // Create the Cosmos DB data client using the account URL and
token credential
            var dataClient = new
CosmosClient($"https://{accountName}.documents.azure.com:443/",
tokenCredential);

            // Create a new database using the management client
            var createDatabaseOperation = await
managementClient.SqlResources.StartCreateUpdateSqlDatabaseAsync(
                resourceGroupName,
                accountName,
                databaseName,
                new SqlDatabaseCreateUpdateParameters(new
SqlDatabaseResource(databaseName), new CreateUpdateOptions()));
            await createDatabaseOperation.WaitForCompletionAsync();

            // Create a new container using the management client
            var createContainerOperation = await
managementClient.SqlResources.StartCreateUpdateSqlContainerAsync(
                resourceGroupName,
                accountName,
                databaseName,
                containerName,
                new SqlContainerCreateUpdateParameters(new
SqlContainerResource(containerName), new CreateUpdateOptions()));
            await createContainerOperation.WaitForCompletionAsync();

            // Create a new item in the container using the data client
            var partitionKey = "pkey";
            var id = Guid.NewGuid().ToString();
            await dataClient.GetContainer(databaseName, containerName)
                .CreateItemAsync(new { id = id, _partitionKey = partitionKey
}, new PartitionKey(partitionKey));

            // Read back the item from the container using the data client
            var pointReadResult = await
dataClient.GetContainer(databaseName, containerName)
                .ReadItemAsync<dynamic>(id, new PartitionKey(partitionKey));

            // Run a query to get all items from the container using the
data client
            await dataClient.GetContainer(databaseName, containerName)
```

```
                .GetItemQueryIterator<dynamic>("SELECT * FROM c")
                .ReadNextAsync();
        }
    }
}
```

Language-specific examples using ManagedIdentityCredential:

.NET

Initialize your Azure Cosmos DB client:

C#

```
CosmosClient client = new CosmosClient("<account-endpoint>", new
ManagedIdentityCredential());
```

Then read and write data.

Java

Initialize your Azure Cosmos DB client:

Java

```
CosmosAsyncClient Client = new CosmosClientBuilder().endpoint("<account-
endpoint>") .credential(new ManagedIdentityCredential()) .build();
```

Then read and write data as described in these samples

JavaScript

Initialize your Azure Cosmos DB client:

JavaScript

```
const client = new CosmosClient({ "<account-endpoint>", aadCredentials: new
ManagedIdentityCredential() });
```

Then read and write data as described in these samples

Clean up steps

Portal

1. Sign in to the Azure portal.
2. Select the resource you want to delete.
3. Select **Delete**.
4. When prompted, confirm the deletion.

Next steps

Learn more about managed identities for Azure resources:

- What are managed identities for Azure resources?
- Azure Resource Manager templates

Learn more about Azure Cosmos DB:

- Azure Cosmos DB resource model
- Tutorial: Build a .NET console app to manage data in an Azure Cosmos DB for NoSQL account

Feedback

Was this page helpful? Yes No

Provide product feedback

Authenticate a managed identity with Microsoft Entra ID to access Azure Service Bus resources

Article • 02/11/2025

Managed identities for Azure resources provide Azure services with an automatically managed identity in Microsoft Entra ID. You can use this identity to authenticate to any service such as Azure Service Bus that supports Microsoft Entra authentication, without having credentials in your code. If you aren't familiar with managed identities, see Managed identities for Azure resources before proceeding to read through this article.

Here are the high-level steps to use a managed identity to access a Service Bus entity:

1. Enable managed identity for your client app or environment. For example, enable managed identity for your Azure App Service app, Azure Functions app, or a virtual machine in which your app is running. Here are the articles that help you with this step:

    - Configure managed identities for App Service and Azure Functions
    - Configure managed identities for Azure resources on a virtual machine (VM)

2. Assign Azure Service Bus Data Owner, Azure Service Bus Data Sender, or Azure Service Bus Data Receiver role to the managed identity at the appropriate scope (Azure subscription, resource group, Service Bus namespace, or Service Bus queue or topic). For instructions to assign a role to a managed identity, see Assign Azure roles using the Azure portal.

3. In your application, use the managed identity and the endpoint to Service Bus namespace to connect to the namespace.

    For example, in .NET, you use the ServiceBusClient constructor that takes `TokenCredential` and `fullyQualifiedNamespace` (a string, for example: `cotosons.servicebus.windows.net`) parameters to connect to Service Bus using the managed identity. You pass in DefaultAzureCredential, which derives from `TokenCredential` and uses the managed identity. In `DefaultAzureCredentialOptions`, set the `ManagedIdentityClientId` to the ID of client's managed identity.

    ```
    C#
    ```

```
string fullyQualifiedNamespace = "<your
namespace>.servicebus.windows.net>";
string userAssignedClientId = "<your managed identity client ID>";

var credential = new DefaultAzureCredential(
    new DefaultAzureCredentialOptions
    {
        ManagedIdentityClientId = userAssignedClientId
    });

var sbusClient = new ServiceBusClient(fullyQualifiedNamespace,
credential);
```

> **Important**
>
> You can disable local or SAS key authentication for a Service Bus namespace and allow only Microsoft Entra authentication. For step-by-step instructions, see Disable local authentication.

Azure built-in roles for Azure Service Bus

Microsoft Entra authorizes access to secured resources through Azure role-based access control (RBAC). Azure Service Bus defines a set of Azure built-in roles that encompass common sets of permissions used to access Service Bus entities. You can also define custom roles for accessing the data.

Azure provides the following Azure built-in roles for authorizing access to a Service Bus namespace:

- Azure Service Bus Data Owner: Use this role to allow full access to Service Bus namespace and its entities (queues, topics, subscriptions, and filters)
- Azure Service Bus Data Sender: Use this role to allow sending messages to Service Bus queues and topics.
- Azure Service Bus Data Receiver: Use this role to allow receiving messages from Service Bus queues and subscriptions.

To assign a role to a managed identity in the Azure portal, use the **Access control (IAM)** page. Navigate to this page by selecting **Access control (IAM)** on the **Service Bus Namespace** page or **Service Bus queue** page, or **Service Bus topic** page. For step-by-step instructions for assigning a role, see Assign Azure roles using the Azure portal.

Resource scope

Before you assign an Azure role to a managed identity, determine the scope of access that the managed identity should have. Best practices dictate that it's always best to grant only the narrowest possible scope.

The following list describes the levels at which you can scope access to Service Bus resources, starting with the narrowest scope:

- **Queue**, **topic**, or **subscription**: Role assignment applies to the specific Service Bus entity.
- **Service Bus namespace**: Role assignment spans the entire topology of Service Bus under the namespace.
- **Resource group**: Role assignment applies to all the Service Bus resources under the resource group.
- **Subscription**: Role assignment applies to all the Service Bus resources in all of the resource groups in the subscription.

> **Note**
>
> Keep in mind that Azure role assignments may take up to five minutes to propagate.

Currently, the Azure portal doesn't support assigning users, groups, or managed identities to Service Bus Azure roles at the topic's subscription level. Here's an example of using the Azure CLI command: az-role-assignment-create to assign an identity to a Service Bus Azure role:

Azure CLI

```
az role assignment create \
    --role $service_bus_role \
    --assignee $assignee_id \
    --scope
/subscriptions/$subscription_id/resourceGroups/$resource_group/providers/Mic
rosoft.ServiceBus/namespaces/$service_bus_namespace/topics/$service_bus_topi
c/subscriptions/$service_bus_subscription
```

For more information about how built-in roles are defined, see Understand role definitions. For information about creating Azure custom roles, see Azure custom roles.

> **Note**

If the source service or app doesn't restart after the access to a Service Bus entity is disabled by removing the source's managed identity from the Service Bus RBAC role, the source app may continue to send/receive messages to/from the Service Bus entity until the token expires (default token validity is 24 hours). This behavior is by design.

Therefore, after you remove the source's managed identity from the RBAC role, restart the source app or service to immediately expire the token and prevent it from sending messages to or receiving messages from the Service Bus entity.

Using SDKs

In .NET, the ServiceBusClient object is initialized by using a constructor that takes a fully qualified namespace and a `TokenCredential`. The `DefaultAzureCredential` derives from `TokenCredential`, which automatically uses the managed identity configured for the app. The flow of the managed identity context to Service Bus and the authorization handshake are automatically handled by the token credential. It's a simpler model than using SAS.

```
C#

var client = new ServiceBusClient('cotosons.servicebus.windows.net', new
DefaultAzureCredential());
```

You send and receive messages as usual using ServiceBusSender and ServiceBusReceiver or ServiceBusProcessor.

For complete step-by-step instructions to send and receive messages using a managed identity, see the following quickstarts. These quickstarts have the code to use a service principal to send and receive messages, but the code is the same for using a managed identity.

- .NET.
- Java.
- JavaScript
- Python

Note

The managed identity works only inside the Azure environment, on App services, Azure VMs, and scale sets. For .NET applications, the

Microsoft.Azure.Services.AppAuthentication library, which is used by the Service Bus NuGet package, provides an abstraction over this protocol and supports a local development experience. This library also allows you to test your code locally on your development machine, using your user account from Visual Studio, Azure CLI 2.0, or Active Directory Integrated Authentication. For more on local development options with this library, see Service-to-service authentication to Azure Key Vault using .NET.

Next steps

See this .NET web application sample on GitHub, which uses a managed identity to connect to Service Bus to send and receive messages. Add the identity of the app service to the **Azure Service Bus Data Owner** role.

Feedback

Use managed identities for App Service and Azure Functions

Article • 03/27/2025

This article shows you how to create a managed identity for Azure App Service and Azure Functions applications, and how to use it to access other resources.

> **Note**
>
> Starting June 1, 2024, newly created App Service apps can generate a unique default host name that uses the naming convention `<app-name>-<random-hash>.<region>.azurewebsites.net`. For example: `myapp-ds27dh7271aah175.westus-01.azurewebsites.net`. Existing app names remain unchanged.
>
> For more information, see the blog post about creating a web app with a unique default host name.

A managed identity from Microsoft Entra ID allows your app to easily access other Microsoft Entra-protected resources, such as Azure Key Vault. The Azure platform manages the identity, so you don't need to provision or rotate any secrets. For more information about managed identities in Microsoft Entra ID, see Managed identities for Azure resources.

You can grant two types of identities to your application:

- A *system-assigned identity* is tied to the app and is deleted if the app is deleted. An app can have only one system-assigned identity.
- A *user-assigned identity* is a standalone Azure resource that can be assigned to your app. An app can have multiple user-assigned identities. One user-assigned identity can be assigned to multiple Azure resources, such as two App Service apps.

The managed identity configuration is specific to the slot. To configure a managed identity for a deployment slot in the portal, go to the slot first. To find the managed identity for your web app or deployment slot in your Microsoft Entra tenant from the Azure portal, search for it directly from the **Overview** page of your tenant. Usually, the slot name is similar to `<app-name>/slots/<slot-name>`.

> **Note**

Managed identities aren't available for apps deployed in Azure Arc.

Because managed identities don't support cross-directory scenarios, they don't behave as expected if your app is migrated across subscriptions or tenants. To re-create the managed identities after such a move, see Will managed identities be re-created automatically if I move a subscription to another directory?. Downstream resources also need to have access policies updated to use the new identity.

Prerequisites

To perform the steps in this article, you must have a minimum set of permissions over your Azure resources. The specific permissions that you need vary based on your scenario. The following table summarizes the most common scenarios:

Expand table

Scenario	Required permission	Example built-in roles
Create a system-assigned identity	`Microsoft.Web/sites/write` over the app, or `Microsoft.Web/sites/slots/write` over the slot	Website Contributor
Create a user-assigned identity	`Microsoft.ManagedIdentity/userAssignedIdentities/write` over the resource group in which to create the identity	Managed Identity Contributor
Assign a user-assigned identity to your app	`Microsoft.Web/sites/write` over the app, `Microsoft.Web/sites/slots/write` over the slot, or `Microsoft.ManagedIdentity/userAssignedIdentities/*/assign/action` over the identity	Website Contributor and Managed Identity Operator
Create Azure role assignments	`Microsoft.Authorization/roleAssignments/write` over the target resource scope	Role Based Access Control Administrator or User Access Administrator

Add a system-assigned identity

To enable a system-assigned managed identity, use the following instructions.

Azure portal

1. In the Azure portal, go to your app's page.
2. On the left menu, select **Settings** > **Identity**.
3. On the **System assigned** tab, switch **Status** to **On**. Then select **Save**.

Add a user-assigned identity

To create an app with a user-assigned identity, create the identity and then add its resource identifier to your app configuration.

Azure portal

1. Create a user-assigned managed identity resource according to these instructions.
2. On the left menu for your app's page, select **Settings** > **Identity**.
3. Select **User assigned**, then select **Add**.
4. Search for the identity that you created earlier, select it, and then select **Add**.

After you finish these steps, the app restarts.

Configure the target resource

You need to configure the target resource to allow access from your app. For most Azure services, you configure the target resource by creating a role assignment.

Some services use mechanisms other than Azure role-based access control. To understand how to configure access by using an identity, refer to the documentation for each target resource. To learn more about which resources support Microsoft Entra tokens, see Azure services that support Microsoft Entra authentication.

For example, if you request a token to access a secret in Azure Key Vault, you must also create a role assignment that allows the managed identity to work with secrets in the target vault. Otherwise, Key Vault rejects your calls even if you use a valid token. The same is true for Azure SQL Database and other services.

> **Important**
>
> The back-end services for managed identities maintain a cache per resource URI for around 24 hours. It can take several hours for changes to a managed identity's group or role membership to take effect. It's currently not possible to force a managed identity's token to be refreshed before its expiration. If you change a managed identity's group or role membership to add or remove permissions, you might need to wait several hours for the Azure resource that's using the identity to have the correct access.
>
> For alternatives to groups or role memberships, see **Limitation of using managed identities for authorization**.

Connect to Azure services in app code

With its managed identity, an app can get tokens for Azure resources that Microsoft Entra ID helps protect, such as Azure SQL Database, Azure Key Vault, and Azure Storage. These tokens represent the application that accesses the resource, and not any specific user of the application.

App Service and Azure Functions provide an internally accessible REST endpoint for token retrieval. You can access the REST endpoint from within the app by using a standard HTTP `GET` request. You can implement the request with a generic HTTP client in every language.

For .NET, JavaScript, Java, and Python, the Azure Identity client library provides an abstraction over this REST endpoint and simplifies the development experience. Connecting to other Azure services is as simple as adding a credential object to the service-specific client.

HTTP GET

A raw HTTP `GET` request uses the two supplied environment variables and looks like the following example:

```
HTTP
```

```
GET /MSI/token?resource=https://vault.azure.net&api-version=2019-08-01
HTTP/1.1
Host: <ip-address-:-port-in-IDENTITY_ENDPOINT>
X-IDENTITY-HEADER: <value-of-IDENTITY_HEADER>
```

A sample response might look like the following example:

HTTP

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "access_token": "eyJ0eXAi…",
    "expires_on": "1586984735",
    "resource": "https://vault.azure.net",
    "token_type": "Bearer",
    "client_id": "00001111-aaaa-2222-bbbb-3333cccc4444"
}
```

This response is the same as the response for the Microsoft Entra service-to-service access token request. To access Key Vault, add the value of `access_token` to a client connection with the vault.

For more information on the REST endpoint, see REST endpoint reference later in this article.

Remove an identity

When you remove a system-assigned identity, it's deleted from Microsoft Entra ID. System-assigned identities are also automatically removed from Microsoft Entra ID when you delete the app resource itself.

Azure portal

1. On the left menu for your app's page, select **Settings** > **Identity**.
2. Follow the steps based on the identity type:
    - For a system-assigned identity: On the **System assigned** tab, switch **Status** to **Off**. Then select **Save**.

- For a user-assigned identity: Select the **User assigned** tab, select the checkbox for the identity, and then select **Remove**. Select **Yes** to confirm.

> **Note**
>
> You can also set an application setting that disables only the local token service: `WEBSITE_DISABLE_MSI`. However, it leaves the identity in place. Tooling still shows the managed identity as on or enabled. As a result, we don't recommend that you use this setting.

REST endpoint reference

An app with a managed identity makes this endpoint available by defining two environment variables:

- `IDENTITY_ENDPOINT`: The URL to the local token service.
- `IDENTITY_HEADER`: A header that can help mitigate server-side request forgery (SSRF) attacks. The platform rotates the value.

The `IDENTITY_ENDPOINT` variable is a local URL from which your app can request tokens. To get a token for a resource, make an HTTP `GET` request to this endpoint. Include the following parameters:

Expand table

Parameter name	In	Description
`resource`	Query	The Microsoft Entra resource URI of the resource for which a token should be obtained. This resource could be one of the Azure services that support Microsoft Entra authentication or any other resource URI.
`api-version`	Query	The version of the token API to be used. Use `2019-08-01`.
`X-IDENTITY-HEADER`	Header	The value of the `IDENTITY_HEADER` environment variable. This header is used to help mitigate SSRF attacks.
`client_id`	Query	(Optional) The client ID of the user-assigned identity to be used. It can't be used on a request that includes `principal_id`, `mi_res_id`, or `object_id`. If all ID parameters (`client_id`, `principal_id`,

Parameter name	In	Description
		`object_id`, and `mi_res_id`) are omitted, the system-assigned identity is used.
`principal_id`	Query	(Optional) The principal ID of the user-assigned identity to be used. The `object_id` parameter is an alias that can be used instead. It can't be used on a request that includes `client_id`, `mi_res_id`, or `object_id`. If all ID parameters (`client_id`, `principal_id`, `object_id`, and `mi_res_id`) are omitted, the system-assigned identity is used.
`mi_res_id`	Query	(Optional) The Azure resource ID of the user-assigned identity to be used. It can't be used on a request that includes `principal_id`, `client_id`, or `object_id`. If all ID parameters (`client_id`, `principal_id`, `object_id`, and `mi_res_id`) are omitted, the system-assigned identity is used.

> **Important**
>
> If you're trying to get tokens for user-assigned identities, include one of the optional properties. Otherwise, the token service tries to get a token for a system-assigned identity, which might or might not exist.

Related content

Consider the following tutorials:

- Connect to SQL Database from .NET App Service without secrets using a managed identity
- Access Azure services from a .NET web app
- Access Microsoft Graph from a secured .NET app as the app
- Secure Cognitive Service connection from .NET App Service using Key Vault

Note: The author created this article with assistance from AI. Learn more

Feedback

Authenticate a managed identity with Microsoft Entra ID to access Event Hubs Resources

Article • 02/11/2025

Azure Event Hubs supports Microsoft Entra authentication with managed identities for Azure resources. Managed identities for Azure resources can authorize access to Event Hubs resources using Microsoft Entra credentials from applications running in Azure Virtual Machines (VMs), Function apps, Virtual Machine Scale Sets, and other services. By using managed identities for Azure resources together with Microsoft Entra authentication, you can avoid storing credentials with your applications that run in the cloud. This article shows how to authorize access to an event hub by using a managed identity from an Azure VM.

Enable managed identities on a VM

Before you use managed identities for Azure resources to access Event Hubs resources from your VM, you must first enable managed identities for Azure Resources on the VM. To learn how to enable managed identities for Azure resources, see Configure managed identities on Azure VMs.

Grant permissions to a managed identity in Microsoft Entra ID

To authorize a request to Event Hubs service from a managed identity in your application, first configure Azure role-based access control (RBAC) settings for that managed identity. Azure Event Hubs defines Azure roles that encompass permissions for sending events to and receiving events from Event Hubs. When an Azure role is assigned to a managed identity, the managed identity is granted access to Event Hubs data at the appropriate scope. For more information about assigning Azure roles, see Authenticate with Microsoft Entra ID for access to Event Hubs resources.

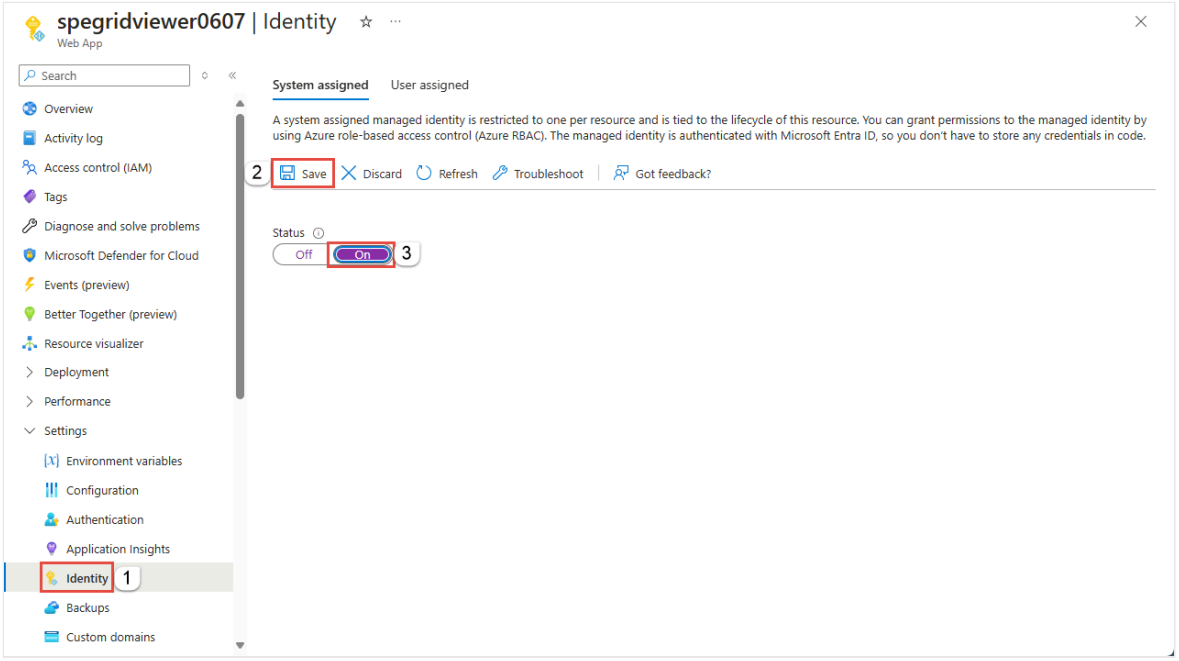
Sample application

The procedure in this section uses a simple application that runs under a managed identity and accesses Event Hubs resources.

Here we're using a sample web application hosted in Azure App Service. For step-by-step instructions for creating a web application, see Create an ASP.NET Core web app in Azure

Once the application is created, follow these steps:

1. Go to **Settings** and select **Identity**.
2. Select the **Status** to be **On**.
3. Select **Save** to save the setting.



4. Select **Yes** on the information message.

   Once you've enabled this setting, a new service identity is created in your Microsoft Entra ID and configured into the App Service host.

   Now, assign this service identity to a role in the required scope in your Event Hubs resources.

To Assign Azure roles using the Azure portal

Assign one of the Event Hubs roles to the managed identity at the desired scope (Event Hubs namespace, resource group, subscription). For detailed steps, see Assign Azure roles using the Azure portal.

> **Note**

For a list of services that support managed identities, see Services that support managed identities for Azure resources.

Test the web application

1. Create an Event Hubs namespace and an event hub.
2. Deploy the web app to Azure. See the following tabbed section for links to the sample web application on GitHub.
3. Ensure that the SendReceive.aspx is set as the default document for the web app.
4. Enable **identity** for the web app.
5. Assign this identity to the **Event Hubs Data Owner** role at the namespace level or event hub level.
6. Run the web application, enter the namespace name and event hub name, a message, and select **Send**. To receive the event, select **Receive**.

You can find the sample web application that sends and receives data from Event Hubs resources in the GitHub repo.

Install the latest package from NuGet, and start sending events to Event Hubs using **EventHubProducerClient** and receiving events using **EventHubConsumerClient**.

> ⓘ Note
>
> For a Java sample that uses a managed identity to publish events to an event hub, see Publish events with Azure identity sample on GitHub.

C#

```
protected async void btnSend_Click(object sender, EventArgs e)
{
    await using (EventHubProducerClient producerClient = new
EventHubProducerClient(txtNamespace.Text, txtEventHub.Text, new
DefaultAzureCredential()))
    {
        // create a batch
        using (EventDataBatch eventBatch = await
producerClient.CreateBatchAsync())
        {

            // add events to the batch. only one in this case.
            eventBatch.TryAdd(new
EventData(Encoding.UTF8.GetBytes(txtData.Text)));

            // send the batch to the event hub
            await producerClient.SendAsync(eventBatch);
```

```
        }

        txtOutput.Text = $"{DateTime.Now} - SENT{Environment.NewLine}
{txtOutput.Text}";
    }
}
protected async void btnReceive_Click(object sender, EventArgs e)
{
    await using (var consumerClient = new
EventHubConsumerClient(EventHubConsumerClient.DefaultConsumerGroupName, $"
{txtNamespace.Text}.servicebus.windows.net", txtEventHub.Text, new
DefaultAzureCredential()))
    {
        int eventsRead = 0;
        try
        {
            using CancellationTokenSource cancellationSource = new
CancellationTokenSource();
            cancellationSource.CancelAfter(TimeSpan.FromSeconds(5));

            await foreach (PartitionEvent partitionEvent in
consumerClient.ReadEventsAsync(cancellationSource.Token))
            {
                txtOutput.Text = $"Event Read: {
Encoding.UTF8.GetString(partitionEvent.Data.Body.ToArray()) }{
Environment.NewLine}" + txtOutput.Text;
                eventsRead++;
            }
        }
        catch (TaskCanceledException ex)
        {
            txtOutput.Text = $"Number of events read: {eventsRead}{
Environment.NewLine}" + txtOutput.Text;
        }
    }
}
```

> ⓘ Note
>
> If the source service or app doesn't restart after the access to the event hub is disabled by removing the source's managed identity from the Event Hubs RBAC role, the source app may continue to publish events to or receiev events from the event hub until the token expires (default token validity is 24 hours). This behavior is by design.
>
> Therefore, after you remove the source's managed identity from the RBAC role, restart the source app or service to immediately expire the token and prevent it from sending events to or receiving events from the event hub.

Event Hubs for Kafka

You can use Apache Kafka applications to send messages to and receive messages from Azure Event Hubs using managed identity OAuth. See the following sample on GitHub: Event Hubs for Kafka - send and receive messages using managed identity OAuth.

Samples

- .NET.
  - For a sample that uses the latest **Azure.Messaging.EventHubs** package, see Publish events with a managed identity
  - For a sample that uses the legacy **Microsoft.Azure.EventHubs** package, see this .NET sample on GitHub
- Java - see the following samples.
  - **Publish events with Azure identity** sample on GitHub.
  - To learn how to use the Apache Kafka protocol to send events to and receive events from an event hub using a managed identity, see Event Hubs for Kafka sample to send and receive messages using a managed identity.

Related content

- See the following article to learn about managed identities for Azure resources: What is managed identities for Azure resources?
- See the following related articles:
  - Authenticate requests to Azure Event Hubs from an application using Microsoft Entra ID
  - Authenticate requests to Azure Event Hubs using Shared Access Signatures
  - Authorize access to Event Hubs resources using Microsoft Entra ID
  - Authorize access to Event Hubs resources using Shared Access Signatures

---

Feedback

How to use managed identities with Azure Container Instances

Article • 08/29/2024

Use managed identities for Azure resources to run code in Azure Container Instances that interacts with other Azure services - without maintaining any secrets or credentials in code. The feature provides an Azure Container Instances deployment with an automatically managed identity in Microsoft Entra ID.

In this article, you learn more about managed identities in Azure Container Instances and:

- ✓ Enable a user-assigned or system-assigned identity in a container group
- ✓ Grant the identity access to an Azure key vault
- ✓ Use the managed identity to access a key vault from a running container

Adapt the examples to enable and use identities in Azure Container Instances to access other Azure services. These examples are interactive. However, in practice your container images would run code to access Azure services.

Why use a managed identity?

Use a managed identity in a running container to authenticate to any service that supports Microsoft Entra authentication without managing credentials in your container code. For services that don't support AD authentication, you can store secrets in an Azure key vault and use the managed identity to access the key vault to retrieve credentials. For more information about using a managed identity, see What is managed identities for Azure resources?

Enable a managed identity

When you create a container group, enable one or more managed identities by setting a ContainerGroupIdentity property. You can also enable or update managed identities after a container group is running - either action causes the container group to restart. To set the identities on a new or existing container group, use the Azure CLI, a Resource Manager template, a YAML file, or another Azure tool.

Azure Container Instances supports both types of managed Azure identities: user-assigned and system-assigned. On a container group, you can enable a system-assigned

identity, one or more user-assigned identities, or both types of identities. If you're unfamiliar with managed identities for Azure resources, see the overview.

Use a managed identity

To use a managed identity, the identity must be granted access to one or more Azure service resources (such as a web app, a key vault, or a storage account) in the subscription. Using a managed identity in a running container is similar to using an identity in an Azure Virtual Machine (VM). See the VM guidance for using a token, Azure PowerShell or Azure CLI, or the Azure SDKs.

Prerequisites

- Use the Bash environment in Azure Cloud Shell. For more information, see Quickstart for Bash in Azure Cloud Shell.

- If you prefer to run CLI reference commands locally, install the Azure CLI. If you're running on Windows or macOS, consider running Azure CLI in a Docker container. For more information, see How to run the Azure CLI in a Docker container.
  - If you're using a local installation, sign in to the Azure CLI by using the az login command. To finish the authentication process, follow the steps displayed in your terminal. For other sign-in options, see Sign in with the Azure CLI.
  - When you're prompted, install the Azure CLI extension on first use. For more information about extensions, see Use extensions with the Azure CLI.
  - Run az version to find the version and dependent libraries that are installed. To upgrade to the latest version, run az upgrade.
- This article requires version 2.0.49 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

Create an Azure key vault

The examples in this article use a managed identity in Azure Container Instances to access an Azure key vault secret.

First, create a resource group named *myResourceGroup* in the *eastus* location with the following az group create command:

Azure CLI

```
az group create --name myResourceGroup --location eastus
```

Use the az keyvault create command to create a key vault. Be sure to specify a unique key vault name.

Azure CLI

```
az keyvault create \
  --name mykeyvault \
  --resource-group myResourceGroup \
  --location eastus
```

Store a sample secret in the key vault using the az keyvault secret set command:

Azure CLI

```
az keyvault secret set \
  --name SampleSecret \
  --value "Hello Container Instances" \
  --description ACIsecret --vault-name mykeyvault
```

Continue with the following examples to access the key vault using either a user-assigned or system-assigned managed identity in Azure Container Instances.

Example 1: Use a user-assigned identity to access Azure key vault

Create an identity

First create an identity in your subscription using the az identity create command. You can use the same resource group used to create the key vault, or use a different one.

Azure CLI

```
az identity create \
  --resource-group myResourceGroup \
  --name myACIId
```

To use the identity in the following steps, use the az identity show command to store the identity's service principal ID and resource ID in variables.

Azure CLI

```
# Get service principal ID of the user-assigned identity
SP_ID=$(az identity show \
  --resource-group myResourceGroup \
  --name myACIId \
  --query principalId --output tsv)

# Get resource ID of the user-assigned identity
RESOURCE_ID=$(az identity show \
  --resource-group myResourceGroup \
  --name myACIId \
  --query id --output tsv)
```

Grant user-assigned identity access to the key vault

Run the following az keyvault set-policy command to set an access policy on the key vault. The following example allows the user-assigned identity to get secrets from the key vault:

Azure CLI

```
az keyvault set-policy \
   --name mykeyvault \
   --resource-group myResourceGroup \
   --object-id $SP_ID \
   --secret-permissions get
```

Enable user-assigned identity on a container group

Run the following az container create command to create a container instance based on Microsoft's `azure-cli` image. This example provides a single-container group that you can use interactively to run the Azure CLI to access other Azure services. In this section, only the base operating system is used. For an example to use the Azure CLI in the container, see Enable system-assigned identity on a container group.

The `--assign-identity` parameter passes your user-assigned managed identity to the group. The long-running command keeps the container running. This example uses the same resource group used to create the key vault, but you could specify a different one.

Azure CLI

```
az container create \
  --resource-group myResourceGroup \
  --name mycontainer \
```

```
  --image mcr.microsoft.com/azure-cli \
  --assign-identity $RESOURCE_ID \
  --command-line "tail -f /dev/null"
```

Within a few seconds, you should get a response from the Azure CLI indicating that the deployment completed. Check its status with the az container show command.

Azure CLI

```
az container show \
  --resource-group myResourceGroup \
  --name mycontainer
```

The `identity` section in the output looks similar to the following, showing the identity is set in the container group. The `principalID` under `userAssignedIdentities` is the service principal of the identity you created in Microsoft Entra ID:

Output

```
[...]
"identity": {
    "principalId": "null",
    "tenantId": "xxxxxxxx-f292-4e60-9122-xxxxxxxxxxxx",
    "type": "UserAssigned",
    "userAssignedIdentities": {
      "/subscriptions/xxxxxxxx-0903-4b79-a55a-
xxxxxxxxxxxx/resourcegroups/danlep1018/providers/Microsoft.ManagedIdentity/u
serAssignedIdentities/myACIId": {
        "clientId": "xxxxxxxx-5523-45fc-9f49-xxxxxxxxxxxx",
        "principalId": "xxxxxxxx-f25b-4895-b828-xxxxxxxxxxxx"
      }
    }
  },
[...]
```

Use user-assigned identity to get secret from key vault

Now you can use the managed identity within the running container instance to access the key vault. First launch a bash shell in the container:

Azure CLI

```
az container exec \
  --resource-group myResourceGroup \
  --name mycontainer \
  --exec-command "/bin/bash"
```

Run the following commands in the bash shell in the container. To get an access token to use Microsoft Entra ID to authenticate to key vault, run the following command:

```Bash
client_id="xxxxxxxx-5523-45fc-9f49-xxxxxxxxxxxx"
curl "http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https%3A%2F%2Fvault.azure.net&client_id=$client_id" -H Metadata:true -s
```

Output:

```Bash
{"access_token":"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Imk2bEdrM0ZaenhSY1ViMkMzbkVRN3N5SEpsWSIsImtpZCI6Imk2bEdrM0ZaenhSY1ViMkMzbkVRN3N5SEpsWSJ9.....xxxxxxxxxxxxxxxx","refresh_token":"","expires_in":"28799","expires_on":"1539927532","not_before":"1539898432","resource":"https://vault.azure.net/","token_type":"Bearer"}
```

For Windows containers, metadata server (169.254.169.254) isn't available. Run the following or equivalent commands to get an access token.

```Console
curl -G -v %IDENTITY_ENDPOINT% --data-urlencode resource=https://vault.azure.net --data-urlencode principalId=<principal id> -H secret:%IDENTITY_HEADER%
```

To store the access token in a variable to use in subsequent commands to authenticate, run the following command:

```Bash
TOKEN=$(curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https%3A%2F%2Fvault.azure.net' -H Metadata:true | jq -r '.access_token')
```

Now use the access token to authenticate to key vault and read a secret. Be sure to substitute the name of your key vault in the URL (*https://mykeyvault.vault.azure.net/...*):

```Bash
```

```
curl https://mykeyvault.vault.azure.net/secrets/SampleSecret/?api-
version=7.4 -H "Authorization: Bearer $TOKEN"
```

The response looks similar to the following, showing the secret. In your code, you would parse this output to obtain the secret. Then, use the secret in a subsequent operation to access another Azure resource.

Bash


```
{"value":"Hello Container
Instances","contentType":"ACIsecret","id":"https://mykeyvault.vault.azure.ne
t/secrets/SampleSecret/xxxxxxxxxxxxxxxxxxxx","attributes":
{"enabled":true,"created":1539965967,"updated":1539965967,"recoveryLevel":"P
urgeable"},"tags":{"file-encoding":"utf-8"}}
```


Example 2: Use a system-assigned identity to access Azure key vault

Enable system-assigned identity on a container group

Run the following az container create command to create a container instance based on Microsoft's `azure-cli` image. This example provides a single-container group that you can use interactively to run the Azure CLI to access other Azure services.

The `--assign-identity` parameter with no additional value enables a system-assigned managed identity on the group. The identity is scoped to the resource group of the container group. The long-running command keeps the container running. This example uses the same resource group used to create the key vault, which is in the scope of the identity.

Azure CLI

```
# Get the resource ID of the resource group
RG_ID=$(az group show --name myResourceGroup --query id --output tsv)

# Create container group with system-managed identity
az container create \
  --resource-group myResourceGroup \
  --name mycontainer \
  --image mcr.microsoft.com/azure-cli \
  --assign-identity --scope $RG_ID \
  --command-line "tail -f /dev/null"
```

Within a few seconds, you should get a response from the Azure CLI indicating that the deployment completed. Check its status with the az container show command.

Azure CLI

```
az container show \
  --resource-group myResourceGroup \
  --name mycontainer
```

The `identity` section in the output looks similar to the following, showing that a system-assigned identity is created in Microsoft Entra ID:

Output

```
[...]
"identity": {
    "principalId": "xxxxxxxx-528d-7083-b74c-xxxxxxxxxxxx",
    "tenantId": "xxxxxxxx-f292-4e60-9122-xxxxxxxxxxxx",
    "type": "SystemAssigned",
    "userAssignedIdentities": null
},
[...]
```

Set a variable to the value of `principalId` (the service principal ID) of the identity, to use in later steps.

Azure CLI

```
SP_ID=$(az container show \
  --resource-group myResourceGroup \
  --name mycontainer \
  --query identity.principalId --out tsv)
```

Grant container group access to the key vault

Run the following az keyvault set-policy command to set an access policy on the key vault. The following example allows the system-managed identity to get secrets from the key vault:

Azure CLI

```
 az keyvault set-policy \
   --name mykeyvault \
   --resource-group myResourceGroup \
```

```
    --object-id $SP_ID \
    --secret-permissions get
```

Use container group identity to get secret from key vault

Now you can use the managed identity to access the key vault within the running container instance. First launch a bash shell in the container:

Azure CLI

```
az container exec \
  --resource-group myResourceGroup \
  --name mycontainer \
  --exec-command "/bin/bash"
```

Run the following commands in the bash shell in the container. First, sign in to the Azure CLI using the managed identity:

Azure CLI

```
az login --identity
```

From the running container, retrieve the secret from the key vault:

Azure CLI

```
az keyvault secret show \
  --name SampleSecret \
  --vault-name mykeyvault --query value
```

The value of the secret is retrieved:

Output

```
"Hello Container Instances"
```

For Windows containers, the 'az login' command won't work because the metadata server is unavailable. Additionally, a managed identity token can't be generated in a Windows virtual network container.

Enable managed identity using Resource Manager template

To enable a managed identity in a container group using a Resource Manager template, set the `identity` property of the `Microsoft.ContainerInstance/containerGroups` object with a `ContainerGroupIdentity` object. The following snippets show the `identity` property configured for different scenarios. See the Resource Manager template reference. Specify a minimum `apiVersion` of `2018-10-01`.

User-assigned identity

A user-assigned identity is a resource ID of the form:

```
"/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{identityName}"
```

You can enable one or more user-assigned identities.

JSON

```
"identity": {
    "type": "UserAssigned",
    "userAssignedIdentities": {
        "myResourceID1": {
            }
        }
    }
```

System-assigned identity

JSON

```
"identity": {
    "type": "SystemAssigned"
    }
```

System- and user-assigned identities

On a container group, you can enable both a system-assigned identity and one or more user-assigned identities.

JSON

```
"identity": {
    "type": "SystemAssigned, UserAssigned",
    "userAssignedIdentities": {
        "myResourceID1": {
            }
        }
    }
...
```

Enable managed identity using YAML file

To enable a managed identity in a container group deployed using a YAML file, include the following YAML. Specify a minimum `apiVersion` of `2018-10-01`.

User-assigned identity

A user-assigned identity is a resource ID of the form

```
'/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{identityName}'
```

You can enable one or more user-assigned identities.

YAML

```yaml
identity:
  type: UserAssigned
  userAssignedIdentities:
    {'myResourceID1':{}}
```

System-assigned identity

YAML

```yaml
identity:
  type: SystemAssigned
```

System- and user-assigned identities

On a container group, you can enable both a system-assigned identity and one or more user-assigned identities.

yml

```yml
identity:
  type: SystemAssigned, UserAssigned
  userAssignedIdentities:
   {'myResourceID1':{}}
```

Next steps

In this article, you learned about managed identities in Azure Container Instances and how to:

- ✓ Enable a user-assigned or system-assigned identity in a container group
- ✓ Grant the identity access to an Azure key vault
- ✓ Use the managed identity to access a key vault from a running container

- Learn more about managed identities for Azure resources.
- See an Azure Go SDK example of using a managed identity to access a key vault from Azure Container Instances.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Connecting from your application to resources without handling credentials

Article • 12/10/2024

Azure resources with managed identities support **always** provide an option to specify a managed identity to connect to Azure resources that support Microsoft Entra authentication. Managed identities support makes it unnecessary for developers to manage credentials in code. Managed identities are the recommended authentication option when working with Azure resources that support them. Read an overview of managed identities.

This page demonstrates how to configure an App Service so it can connect to Azure Key Vault, Azure Storage, and Microsoft SQL Server. The same principles can be used for any Azure resource that supports managed identities and that will connect to resources that support Microsoft Entra authentication.

The code samples use the Azure Identity client library, which is the recommended method as it automatically handles many of the steps for you, including acquiring an access token used in the connection.

What resources can managed identities connect to?

A managed identity can connect to any resource that supports Microsoft Entra authentication. In general, there's no special support required for the resource to allow managed identities to connect to it.

Some resources don't support Microsoft Entra authentication, or their client library doesn't support authenticating with a token. Keep reading to see our guidance on how to use a Managed identity to securely access the credentials without needing to store them in your code or application configuration.

Creating a managed identity

There are two types of managed identities: system-assigned and user-assigned. System-assigned identities are directly linked to a single Azure resource. When the Azure resource is deleted, so is the identity. A user-assigned managed identity can be associated with multiple Azure resources, and its lifecycle is independent of those resources.

We recommend that you use a user-assigned managed identity, for most scenarios. If the source resource you're using doesn't support user-assigned managed identities, then you should refer to that resource provider's documentation to learn how to configure it to have a system-assigned managed identity.

> **Important**
>
> The account used to create managed identities needs a role such as "Managed Identity Contributor" to create a new user-assigned managed identity.

Create a user-assigned managed identity using your preferred option:

- Azure portal
- Azure CLI
- Azure PowerShell
- Resource Manager
- REST

After you create a user-assigned managed identity, take note of the `clientId` and the `principalId` values that are returned when the managed identity is created. You use `principalId` while adding permissions, and `clientId` in your application's code.

Configure App Service with a user-assigned managed identity

Before you can use the managed identity in your code, we have to assign it to the App Service that will use it. The process of configuring an App Service to use a user-assigned managed identity requires that you specify the managed identity's resource identifier in your app config.

Adding permissions to the identity

Once you've configured your App Service to use a user-assigned managed identity, grant the necessary permissions to the identity. In this scenario, we're using this identity to interact with Azure Storage, so you need to use the Azure Role Based Access Control (RBAC) system to grant the user-assigned managed identity permissions to the resource.

> **Important**

You'll need a role such as "User Access Administrator" or "Owner" for the target resource to add Role assignments. Ensure you're granting the least privilege required for the application to run.

Any resources you want to access requires that you grant the identity permissions. For example, if you request a token to access Key Vault, you must also add an access policy that includes the managed identity of your app or function. Otherwise, your calls to Key Vault will be rejected, even if you use a valid token. The same is true for Azure SQL Database. To learn more about which resources support Microsoft Entra tokens, see Azure services that support Microsoft Entra authentication.

Using managed identities in your code

After you complete the steps outlined above, your App Service has a managed identity with permissions to an Azure resource. You can use the managed identity to obtain an access token that your code can use to interact with Azure resources, instead of storing credentials in your code.

We recommended that you use the client libraries that we provide for your preferred programming language. These libraries acquire access tokens for you, making it easy to authenticate with Microsoft Entra ID. For more information, see Client libraries for managed identities authentication.

Using an Azure Identity library to access Azure resources

The Azure Identity libraries each provide a `DefaultAzureCredential` type. `DefaultAzureCredential` attempts to automatically authenticate the user through different flows, including environment variables or an interactive sign-in. The credential type can be used in a development environment with your own credentials. It can also be used in your production Azure environment using a managed identity. No code changes are required when you deploy your application.

If you're using user-assigned managed identities, you should also explicitly specify the user-assigned managed identity you wish to authenticate with by passing in the identity's client ID as a parameter. You can retrieve the client ID by browsing to the identity in the Azure portal.

Accessing a Blob in Azure Storage

.NET

C#

```
using Azure.Identity;
using Azure.Storage.Blobs;

// code omitted for brevity

// Specify the Client ID if using user-assigned managed identities
var clientID =
Environment.GetEnvironmentVariable("Managed_Identity_Client_ID");
var credentialOptions = new DefaultAzureCredentialOptions
{
    ManagedIdentityClientId = clientID
};
var credential = new DefaultAzureCredential(credentialOptions);

var blobServiceClient1 = new BlobServiceClient(new Uri("<URI of Storage
account>"), credential);
BlobContainerClient containerClient1 =
blobServiceClient1.GetBlobContainerClient("<name of blob>");
BlobClient blobClient1 = containerClient1.GetBlobClient("<name of
file>");

if (blobClient1.Exists())
{
    var downloadedBlob = blobClient1.Download();
    string blobContents = downloadedBlob.Value.Content.ToString();
}
```

Accessing a secret stored in Azure Key Vault

.NET

C#

```
using Azure.Identity;
using Azure.Security.KeyVault.Secrets;
using Azure.Core;

// code omitted for brevity

// Specify the Client ID if using user-assigned managed identities
var clientID =
Environment.GetEnvironmentVariable("Managed_Identity_Client_ID");
var credentialOptions = new DefaultAzureCredentialOptions
{
    ManagedIdentityClientId = clientID
};
var credential = new DefaultAzureCredential(credentialOptions);
```

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

KeyVaultSecret secret = client.GetSecret("<my secret>");
string secretValue = secret.Value;
```

Accessing Azure SQL Database

.NET

C#

```
using Azure.Identity;
using Microsoft.Data.SqlClient;

// code omitted for brevity

// Specify the Client ID if using user-assigned managed identities
var clientID =
Environment.GetEnvironmentVariable("Managed_Identity_Client_ID");
var credentialOptions = new DefaultAzureCredentialOptions
{
    ManagedIdentityClientId = clientID
};

AccessToken accessToken = await new
DefaultAzureCredential(credentialOptions).GetTokenAsync(
    new TokenRequestContext(new string[] {
"https://database.windows.net//.default" }));

using var connection = new SqlConnection("Server=<DB Server>; Database=
<DB Name>;")
{
    AccessToken = accessToken.Token
};
var cmd = new SqlCommand("select top 1 ColumnName from TableName",
connection);
await connection.OpenAsync();
SqlDataReader dr = cmd.ExecuteReader();
while(dr.Read())
{
    Console.WriteLine(dr.GetValue(0).ToString());
}
dr.Close();
```

Using the Microsoft Authentication Library (MSAL) to access Azure resources

Apart from the Azure Identity libraries, you can also use MSAL to access Azure resources using managed identities. The following code snippets demonstrate how to use MSAL to access Azure resources in various programming languages.

For system-assigned managed identities, the developer doesn't need to pass any additional information. MSAL automatically infers the relevant metadata about the assigned identity. For user-assigned managed identities, the developer needs to pass either the client ID, full resource identifier, or the object ID of the managed identity.

You can then acquire a token to access a resource. Prior to using managed identities, developers must enable them for the resources they want to use.

.NET

C#

```
using Microsoft.Identity.Client;
using System;

string resource = "https://vault.azure.net";

// Applies to system-assigned managed identities only
IManagedIdentityApplication mi =
ManagedIdentityApplicationBuilder.Create(ManagedIdentityId.SystemAssigne
d)
    .Build();

// Applies to user-assigned managed identities only
string userAssignedManagedIdentityClientId = "xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxxx";
IManagedIdentityApplication mi =
ManagedIdentityApplicationBuilder.Create(ManagedIdentityId.WithUserAssig
nedClientId(userAssignedManagedIdentityClientId))
    .Build();

// Acquire token
AuthenticationResult result = await
mi.AcquireTokenForManagedIdentity(resource)
    .ExecuteAsync()
    .ConfigureAwait(false);

if (!string.IsNullOrEmpty(result.AccessToken))
{
    Console.WriteLine(result.AccessToken);
```

```
}
```

Connecting to resources that don't support Microsoft Entra ID or token based authentication in libraries

Some Azure resources either don't yet support Microsoft Entra authentication, or their client libraries don't support authenticating with a token. Typically these resources are open-source technologies that expect a username and password or an access key in a connection string.

To avoid storing credentials in your code or your application configuration, you can store the credentials as a secret in Azure Key Vault. Using the example displayed above, you can retrieve the secret from Azure KeyVault using a managed identity, and pass the credentials into your connection string. This approach means that no credentials need to be handled directly in your code or environment. For a detailed example, see Use managed identities to access Azure Key Vault certificates. For more info on Azure Key Vault authentication, see Azure Key Vault authentication.

Guidelines if you're handling tokens directly

In some scenarios, you may want to acquire tokens for managed identities manually instead of using a built-in method to connect to the target resource. These scenarios include no client library for the programming language that you're using or the target resource you're connecting to, or connecting to resources that aren't running on Azure. When acquiring tokens manually, we provide the following guidelines:

Cache the tokens you acquire

For performance and reliability, we recommend that your application caches tokens in local memory, or encrypted if you want to save them to disk. As Managed identity tokens are valid for 24 hours, there's no benefit in requesting new tokens regularly, as a cached one will be returned from the token issuing endpoint. If you exceed the request limits, you'll be rate limited and receive an HTTP 429 error.

When you acquire a token, you can set your token cache to expire 5 minutes before the `expires_on` (or equivalent property) that will be returned when the token is generated.

Token inspection

Your application shouldn't rely on the contents of a token. The token's content is intended only for the audience (target resource) that is being accessed, not the client that's requesting the token. The token content may change or be encrypted in the future.

Don't expose or move tokens

Tokens should be treated like credentials. Don't expose them to users or other services; for example, logging/monitoring solutions. They shouldn't be moved from the source resource that's using them, other than to authenticate against the target resource.

Next steps

- How to use managed identities for App Service and Azure Functions
- How to use managed identities with Azure Container Instances
- Implementing managed identities for Microsoft Azure Resources
- Use workload identity federation for managed identities to access Microsoft Entra protected resources without managing secrets

ⓘ **Note:** The author created this article with assistance from AI. Learn more

Feedback

Was this page helpful? Yes No

Provide product feedback

How managed identities for Azure resources work with Azure virtual machines

Article • 03/14/2025

Managed identities for Azure resources provide Azure services with an automatically managed identity in Microsoft Entra ID. You can use this identity to authenticate to any service that supports Microsoft Entra authentication, without having credentials in your code.

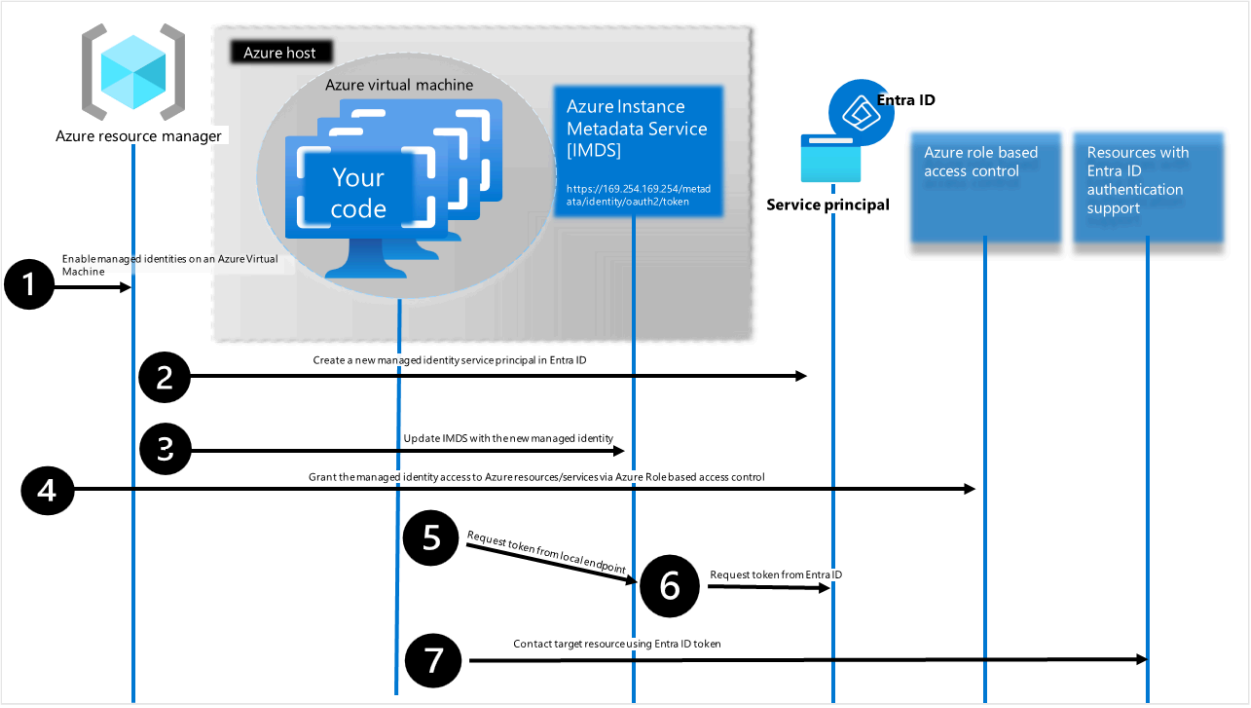
In this article, you learn how managed identities work with Azure virtual machines (VMs).

How it works

Internally, managed identities are service principals of a special type, which can only be used with Azure resources. When the managed identity is deleted, the corresponding service principal is automatically removed. Also, when a User-Assigned or System-Assigned Identity is created, the Managed Identity Resource Provider (MSRP) issues a certificate internally to that identity.

Your code can use a managed identity to request access tokens for services that support Microsoft Entra authentication. Azure takes care of rolling the credentials that are used by the service instance.

The following diagram shows how managed service identities work with Azure virtual machines (VMs):



The following table shows the differences between the system-assigned and user-assigned managed identities:

Expand table

Property	System-assigned managed identity	User-assigned managed identity
Creation	Created as part of an Azure resource (for example, an Azure virtual machine or Azure App Service).	Created as a stand-alone Azure resource.
Life cycle	Shared life cycle with the Azure resource that the managed identity is created with. When the parent resource is deleted, the managed identity is deleted as well.	Independent life cycle. Must be explicitly deleted.
Sharing across Azure resources	Can't be shared. It can only be associated with a single Azure resource.	Can be shared. The same user-assigned managed identity can be associated with more than one Azure resource.
Common use cases	Workloads that are contained within a single Azure resource. Workloads for which you need independent identities. For example, an application that runs on a single virtual machine	Workloads that run on multiple resources and which can share a single identity. Workloads that need pre-authorization to a secure resource as part of a provisioning flow. Workloads where resources are recycled frequently, but permissions should stay consistent.

Property	System-assigned managed identity	User-assigned managed identity
		For example, a workload where multiple virtual machines need to access the same resource

System-assigned managed identity

1. Azure Resource Manager receives a request to enable the system-assigned managed identity on a VM.

2. Azure Resource Manager creates a service principal in Microsoft Entra ID for the identity of the VM. The service principal is created in the subscription's trusted Microsoft Entra tenant.

3. Azure Resource Manager updates the VM identity using the Azure Instance Metadata Service identity endpoint (for Windows and Linux), providing the endpoint with the service principal client ID and certificate.

4. After the VM has an identity, use the service principal information to grant the VM access to Azure resources. To call Azure Resource Manager, use Azure Role-Based Access Control (Azure RBAC) to assign the appropriate role to the VM service principal. To call Key Vault, grant your code access to the specific secret or key in Key Vault.

5. Your code that's running on the VM can request a token from the Azure Instance Metadata service endpoint, accessible only from within the VM: `http://169.254.169.254/metadata/identity/oauth2/token`

   - The resource parameter specifies the service to which the token is sent. To authenticate to Azure Resource Manager, use `resource=https://management.azure.com/`.
   - API version parameter specifies the IMDS version. Use api-version=2018-02-01 or later.

   The following example demonstrates how to to use CURL to make a request to the local Managed Identity endpoint to get an access token for Azure Instance Metadata service.

   Bash

   ```
   curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https%3A%2F%2Fstorage.azure.com%2F' -H
   ```

```
Metadata:true
```

6. A call is made to Microsoft Entra ID to request an access token (as specified in step 5) by using the client ID and certificate configured in step 3. Microsoft Entra ID returns a JSON Web Token (JWT) access token.

7. Your code sends the access token on a call to a service that supports Microsoft Entra authentication.

User-assigned managed identity

1. Azure Resource Manager receives a request to create a user-assigned managed identity.

2. Azure Resource Manager creates a service principal in Microsoft Entra ID for the user-assigned managed identity. The service principal is created in the subscription's trusted Microsoft Entra tenant.

3. Azure Resource Manager receives a request to configure the user-assigned managed identity on a VM and updates the Azure Instance Metadata Service identity endpoint with the user-assigned managed identity service principal client ID and certificate.

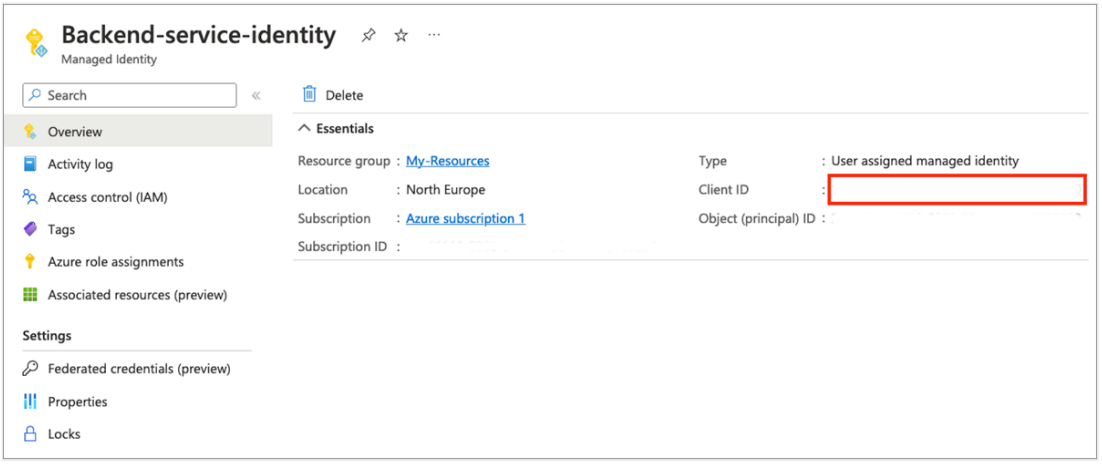
4. After the user-assigned managed identity is created, use the service principal information to grant the identity access to Azure resources. To call Azure Resource Manager, use Azure RBAC to assign the appropriate role to the service principal of the user-assigned identity. To call Key Vault, grant your code access to the specific secret or key in Key Vault.

   > **Note**
   >
   > You can also do this step before step 3.

5. Your code that's running on the VM can request a token from the Azure Instance Metadata Service identity endpoint, accessible only from within the VM: `http://169.254.169.254/metadata/identity/oauth2/token`

   - The resource parameter specifies the service to which the token is sent. To authenticate to Azure Resource Manager, use `resource=https://management.azure.com/`.

- The `client_id` parameter specifies the identity for which the token is requested. This value is required for disambiguation when more than one user-assigned identity is on a single VM. You can find the **Client ID** in the Managed Identity **Overview**:



- The API version parameter specifies the Azure Instance Metadata Service version. Use `api-version=2018-02-01` or higher.

  The following example demonstrates how to to use CURL to make a request to the local Managed Identity endpoint to get an access token for Azure Instance Metadata service.

  Bash

  ```
  curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https%3A%2F%2Fstorage.azure.com%2F&client_id=00001111-aaaa-2222-bbbb-3333cccc4444' -H Metadata:true
  ```

6. A call is made to Microsoft Entra ID to request an access token (as specified in step 5) by using the client ID and certificate configured in step 3. Microsoft Entra ID returns a JSON Web Token (JWT) access token.

7. Your code sends the access token on a call to a service that supports Microsoft Entra authentication.

Next steps

Get started with the managed identities for Azure resources feature with the following quickstarts:

- Use a Windows VM system-assigned managed identity to access Resource Manager

- Use a Linux VM system-assigned managed identity to access Resource Manager

Feedback

Was this page helpful? Yes No

Provide product feedback

Managed identity best practice recommendations

Article • 03/14/2025

Managed identities in Azure provide a secure and convenient way to manage credentials for applications running on Azure resources. This article outlines best practice recommendations for choosing between user-assigned and system-assigned managed identities, helping you optimize identity management and reduce administrative overhead.

Choosing system or user-assigned managed identities

User-assigned managed identities are more efficient in a broader range of scenarios than system-assigned managed identities. See the following table for some scenarios and the recommendations for user-assigned or system-assigned.

User-assigned identities can be used by multiple resources, and their life cycles are decoupled from the resources' life cycles with which they're associated. Read which resources support managed identities.

This life cycle allows you to separate your resource creation and identity administration responsibilities. User-assigned identities and their role assignments can be configured in advance of the resources that require them. Users who create the resources only require the access to assign a user-assigned identity, without the need to create new identities or role assignments.

As system-assigned identities are created and deleted along with the resource, role assignments can't be created in advance. This sequence can cause failures while deploying infrastructure if the user creating the resource doesn't also have access to create role assignments.

If your infrastructure requires that multiple resources require access to the same resources, a single user-assigned identity can be assigned to them. Administration overhead is reduced, as there are fewer distinct identities and role assignments to manage.

If you require that each resource has its own identity, or have resources that require a unique set of permissions and want the identity to be deleted as the resource is deleted, then you should use a system-assigned identity.

Expand table

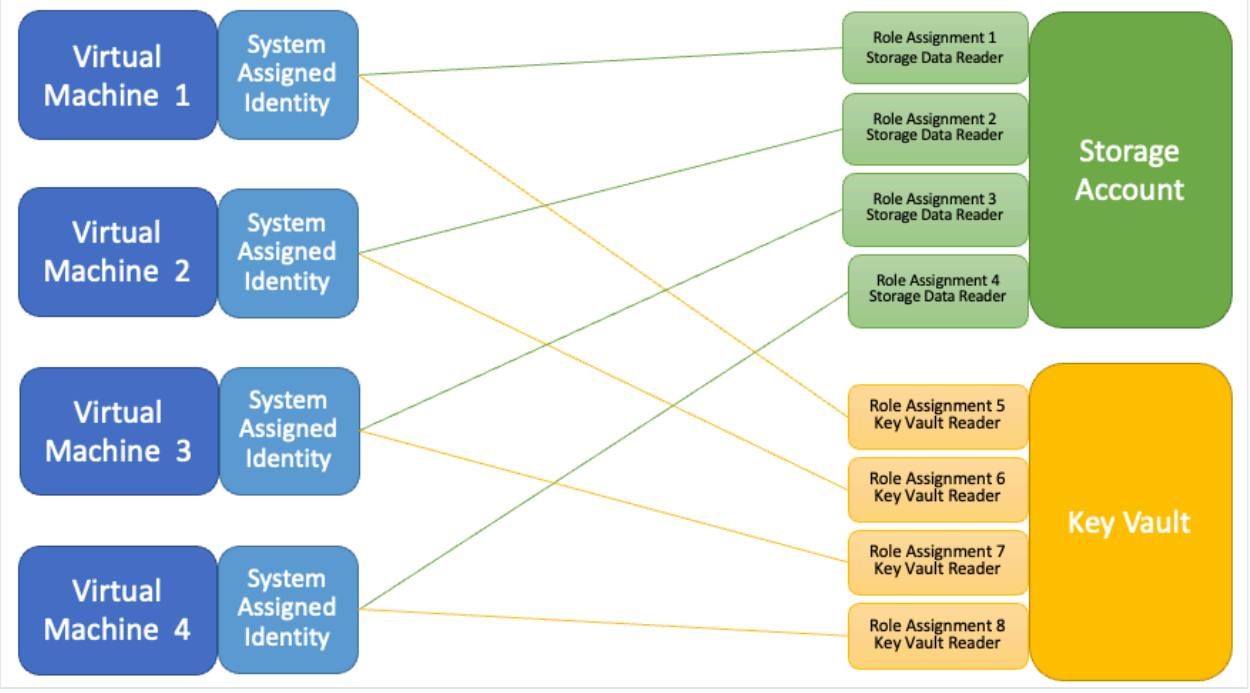
Scenario	Recommendation	Notes
Rapid creation of resources (for example, ephemeral computing) with managed identities	User-assigned identity	If you attempt to create multiple managed identities in a short space of time – for example, deploying multiple virtual machines each with their own system-assigned identity - you may exceed the rate limit for Microsoft Entra object creations, and the request fails with an HTTP 429 error. If resources are being created or deleted rapidly, you may also exceed the limit on the number of resources in Microsoft Entra ID if using system-assigned identities. While a deleted system-assigned identity is no longer accessible by any resource, it counts towards your limit until fully purged after 30 days. Deploying the resources associated with a single user-assigned identity require the creation of only one Service Principal in Microsoft Entra ID, avoiding the rate limit. Using a single identity that is created in advance reduces the risk of replication delays that could occur if multiple resources are created each with their own identity. Read more about the Azure subscription service limits.
Replicated resources/applications	User-assigned identity	Resources that carry out the same task – for example, duplicated web servers or identical functionality running in an app service and in an application on a virtual machine – typically require the same permissions. By using the same user-assigned identity, fewer role assignments are required which reduces the management overhead. The resources don't have to be of the same type.
Compliance	User-assigned identity	If your organization requires that all identity creation must go through an approval process, using a single user-assigned identity across multiple resources requires fewer approvals than system-assigned

Scenario	Recommendation	Notes
		Identities, which are created as new resources are created.
Access required before a resource is deployed	User-assigned identity	Some resources may require access to certain Azure resources as part of their deployment. In this case, a system-assigned identity may not be created in time so a preexisting user-assigned identity should be used.
Audit Logging	System-assigned identity	If you need to log which specific resource carried out an action, rather than which identity, use a system-assigned identity.
Permissions Lifecycle Management	System-assigned identity	If you require that the permissions for a resource be removed along with the resource, use a system-assigned identity.

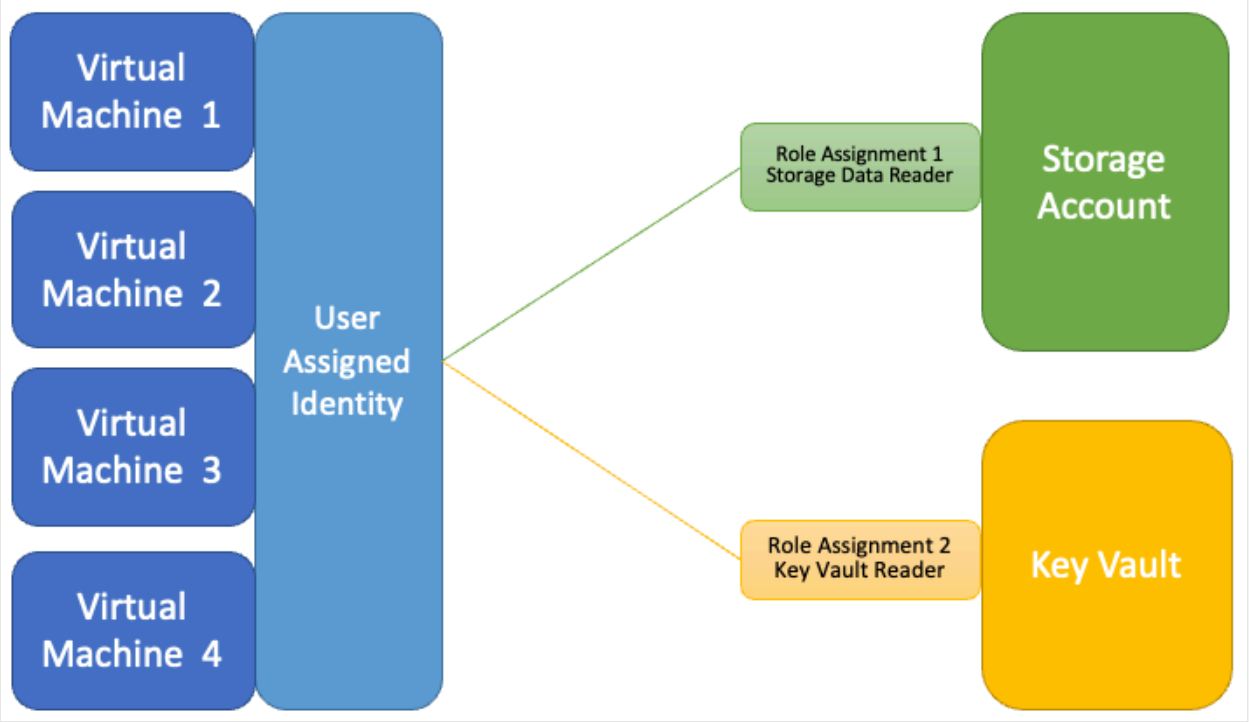
Using user-assigned identities to reduce administration

The diagrams demonstrate the difference between system-assigned and user-assigned identities, when used to allow several virtual machines to access two storage accounts.

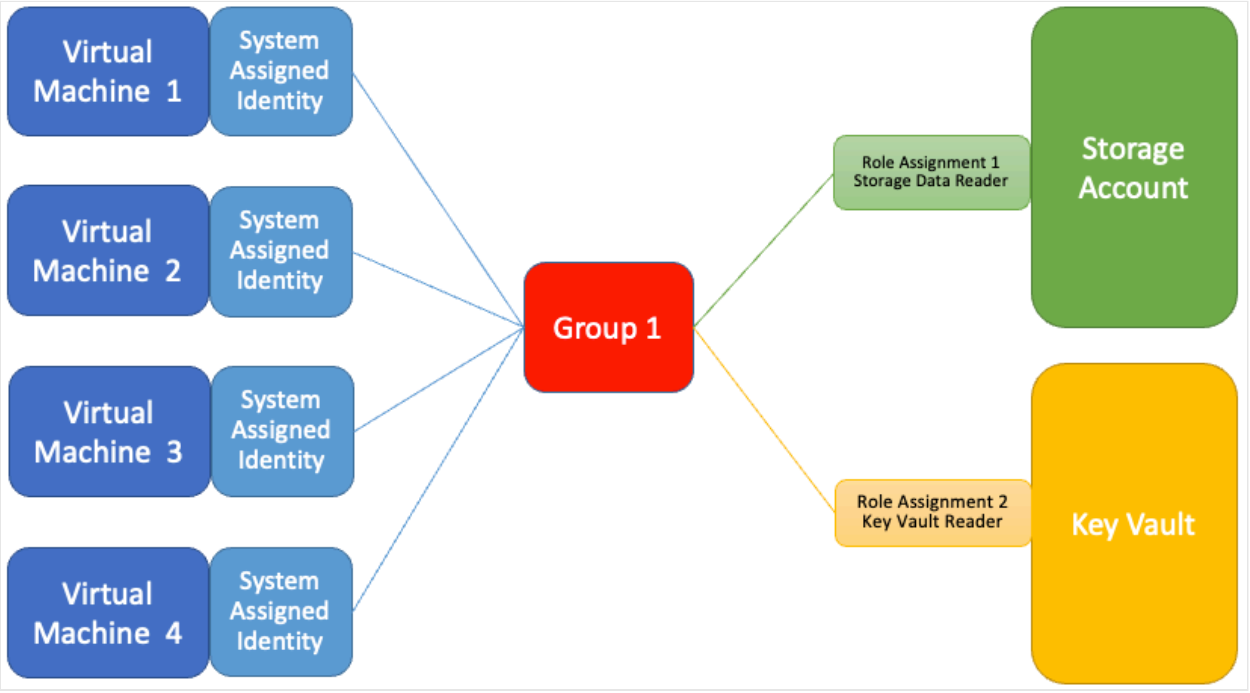
The diagram shows four virtual machines with system-assigned identities. Each virtual machine has the same role assignments that grants them access to two storage accounts.



When a user-assigned identity is associated with the four virtual machines, only two role assignments are required, compared to eight with system-assigned identities. If the virtual machines' identity requires more role assignments, they are granted to all the resources associated with this identity.



Security groups can also be used to reduce the number of role assignments that are required. This diagram shows four virtual machines with system-assigned identities, which were added to a security group, with the role assignments added to the group instead of the system-assigned identities. While the result is similar, this configuration doesn't offer the same Resource Manager template capabilities as user-assigned identities.

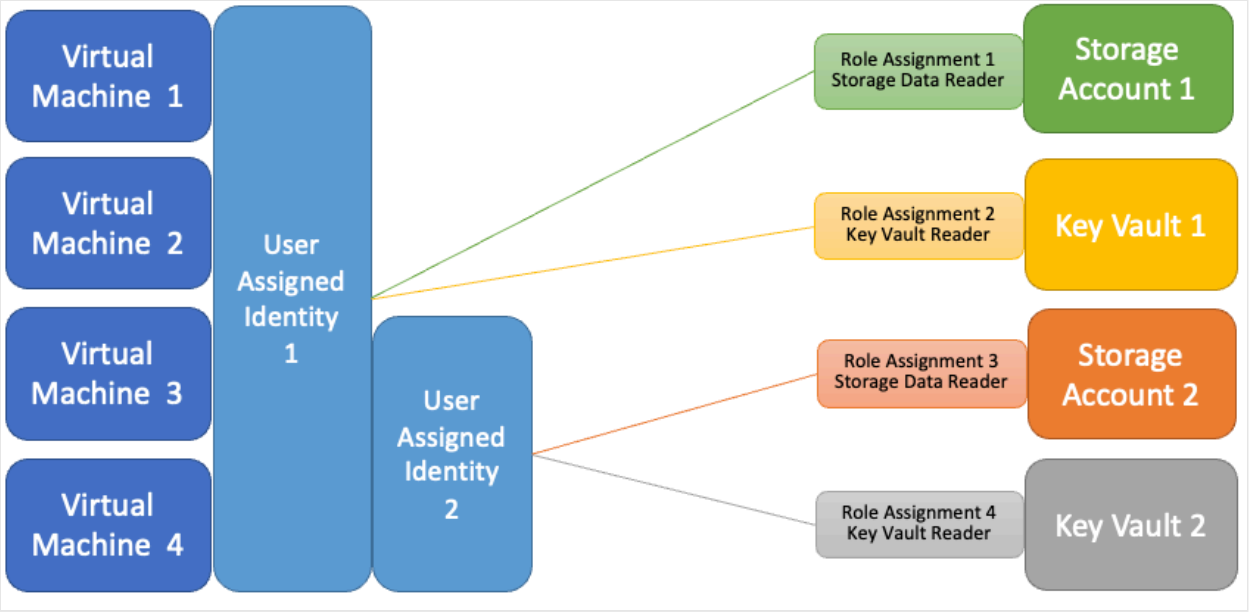


Multiple managed identities

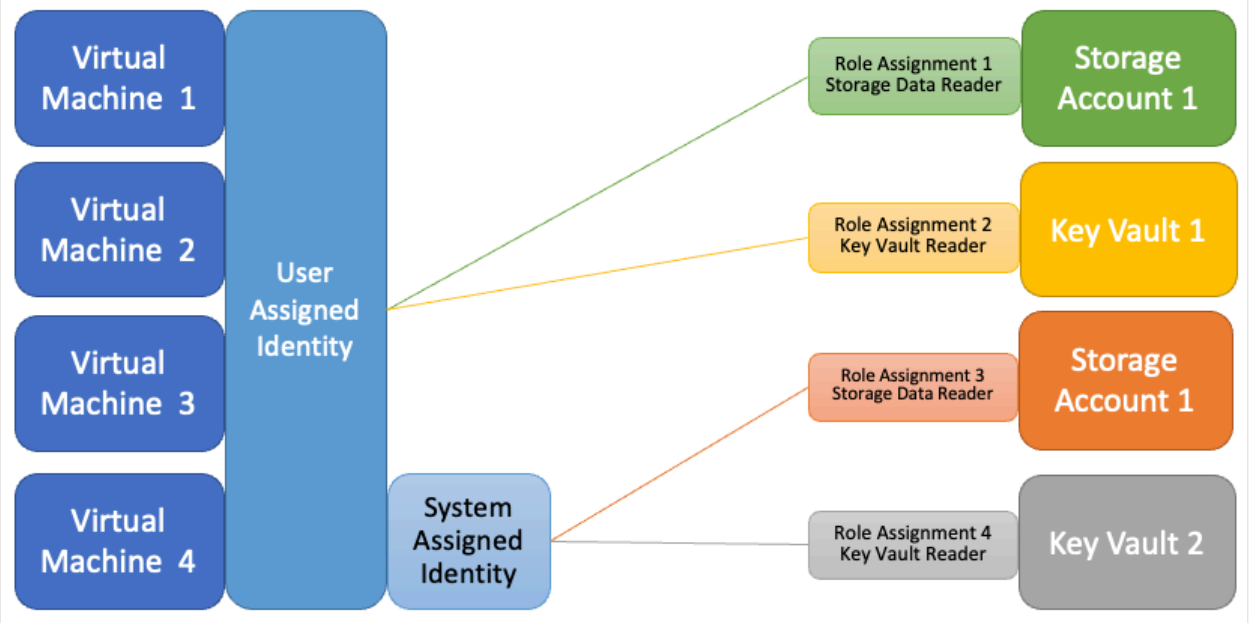
Resources that support managed identities can have both a system-assigned identity and one or more user-assigned identities.

This model provides the flexibility to both use a shared user-assigned identity and apply granular permissions when needed.

In the following example, "Virtual Machine 3" and "Virtual Machine 4" can access both storage accounts and key vaults, depending on which user-assigned identity they use while authenticating.



In the following example, "Virtual Machine 4" has both a user-assigned identity, giving it access to both storage accounts and key vaults, depending on which identity is used while authenticating. The role assignments for the system-assigned identity are specific to that virtual machine.



Limits

View the limits for managed identities and for custom roles and role assignments.

Follow the principle of least privilege when granting access

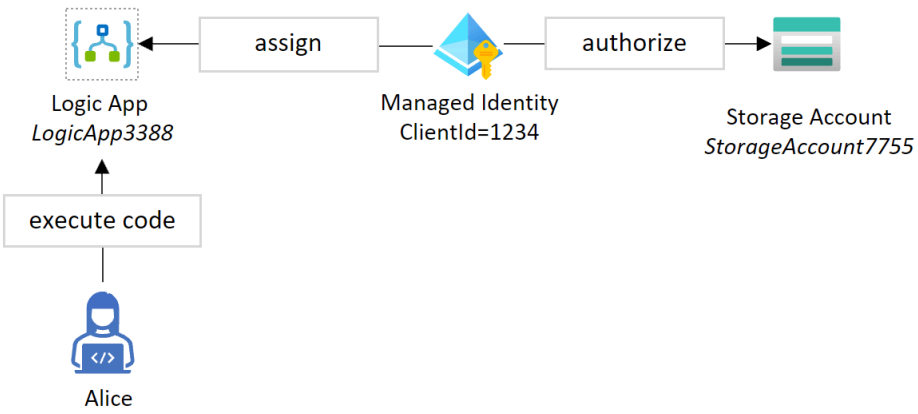
When granting any identity, including a managed identity, permissions to access services, always grant the least permissions needed to perform the desired actions. For example, if a managed identity is used to read data from a storage account, there's no need to allow that identity permissions to also write data to the storage account. Granting extra permissions, for example, making the managed identity a contributor on an Azure subscription when it's not needed, increases the security blast radius associated with the identity. One must always minimize the security blast radius so that compromising that identity causes minimum damage.

Consider the effect of assigning managed identities to Azure resources and/or granting assign permissions to a user

It's important to note that when an Azure resource, such as an Azure Logic App or a Virtual Machine, is assigned a managed identity, all the permissions granted to the managed identity are now available to the Azure resource. This is important because if a user has access to install or execute code on this resource, then the user has access to all the identities assigned/associated to the Azure resource. The purpose of managed identity is to give code running on an Azure resource access to other resources, without developers needing to handle or put credentials directly into code to get that access.

For example, if a managed Identity (ClientId = 1234) is granted read/write access to ***StorageAccount7755*** and is assigned to ***LogicApp3388***, then Alice, who doesn't have direct access to the storage account but has permission to execute code within ***LogicApp3388*** can also read/write data to/from ***StorageAccount7755*** by executing the code that uses the managed identity.

Similarly, if Alice has permissions to assign the managed identity herself, she can assign it to a different Azure resource and have access to all the permissions available to the managed identity.



In general, when granting a user administrative access to a resource that can execute code (such as a Logic App) and has a managed identity, consider if the role being assigned to the user can install or run code on the resource, and if yes only assign that role if the user really needs it.

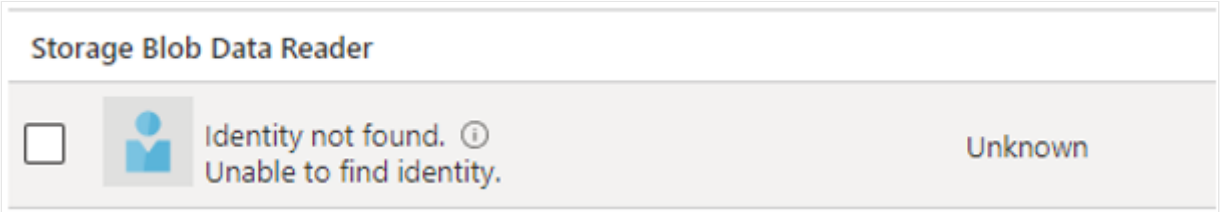
Maintenance

System-assigned identities are automatically deleted when the resource is deleted, while the lifecycle of a user-assigned identity is independent of any resources with which it's associated.

You need to manually delete a user-assigned identity when it's no longer required, even if no resources are associated with it.

Role assignments aren't automatically deleted when either system-assigned or user-assigned managed identities are deleted. These role assignments should be manually deleted so the limit of role assignments per subscription isn't exceeded.

Role assignments that are associated with deleted managed identities are displayed with "Identity not found" when viewed in the portal. Read more.



Role assignments which are no longer associated with a user or service principal appears with an `ObjectType` value of `Unknown`. In order to remove them, you can pipe several Azure PowerShell commands together to first get all the role assignments, filter to only those with an `ObjectType` value of `Unknown` and then remove those role assignments from Azure.

Azure PowerShell

```
Get-AzRoleAssignment | Where-Object {$_.ObjectType -eq "Unknown"} | Remove-AzRoleAssignment
```

Limitation of using managed identities for authorization

Using Microsoft Entra ID **groups** for granting access to services is a great way to simplify the authorization process. The idea is simple – grant permissions to a group and add identities to the group so that they inherit the same permissions. This is a well-established pattern from various on-premises systems and works well when the identities represent users. Another option to control authorization in Microsoft Entra ID is by using App Roles, which allows you to declare **roles** that are specific to an app (rather than groups, which are a global concept in the directory). You can then assign app roles to managed identities (as well as users or groups).

In both cases, for non-human identities such as Microsoft Entra Applications and Managed identities, the exact mechanism of how this authorization information is presented to the application isn't ideally suited today. Today's implementation with Microsoft Entra ID and Azure Role Based Access Control (Azure RBAC) uses access tokens issued by Microsoft Entra ID for authentication of each identity. If the identity is added to a group or role, this is expressed as claims in the access token issued by Microsoft Entra ID. Azure RBAC uses these claims to further evaluate the authorization rules for allowing or denying access.

Given that the identity's groups and roles are claims in the access token, any authorization changes don't take effect until the token is refreshed. For a human user that's typically not a problem, because a user can acquire a new access token by logging out and in again (or waiting for the token lifetime to expire, which is 1 hour by default). Managed identity tokens on the other hand are cached by the underlying Azure infrastructure for performance and resiliency purposes: the back-end services for managed identities maintain a cache per resource URI for around 24 hours. This means that it can take several hours for changes to a managed identity's group or role membership to take effect. Today, it isn't possible to force a managed identity's token to

be refreshed before its expiry. If you change a managed identity's group or role membership to add or remove permissions, you may therefore need to wait several hours for the Azure resource using the identity to have the correct access.

If this delay isn't acceptable for your requirements, consider alternatives to using groups or roles in the token. To ensure that changes to permissions for managed identities take effect quickly, we recommend that you group Azure resources using a user-assigned managed identity with permissions applied directly to the identity, instead of adding to or removing managed identities from a Microsoft Entra group that has permissions. A user-assigned managed identity can be used like a group because it can be assigned to one or more Azure resources to use it. The assignment operation can be controlled using the Managed identity contributor and Managed identity operator role.

Feedback

Configure managed identities on Azure virtual machines (VMs)

Article • 01/16/2025

Managed identities for Azure resources is a feature of Microsoft Entra ID. Each of the Azure services that support managed identities for Azure resources are subject to their own timeline. Make sure you review the availability status of managed identities for your resource and known issues before you begin.

Managed identities for Azure resources provide Azure services with an automatically managed identity in Microsoft Entra ID. You can use this identity to authenticate to any service that supports Microsoft Entra authentication, without having credentials in your code.

For information about Azure Policy definition and details, see Use Azure Policy to assign managed identities (preview).

In this article, you learn how to enable and disable system and user-assigned managed identities for an Azure Virtual Machine (VM), using the Azure portal.

Prerequisites

- If you're unfamiliar with managed identities for Azure resources, check out the overview section.
- If you don't already have an Azure account, sign up for a free account before continuing.

System-assigned managed identity

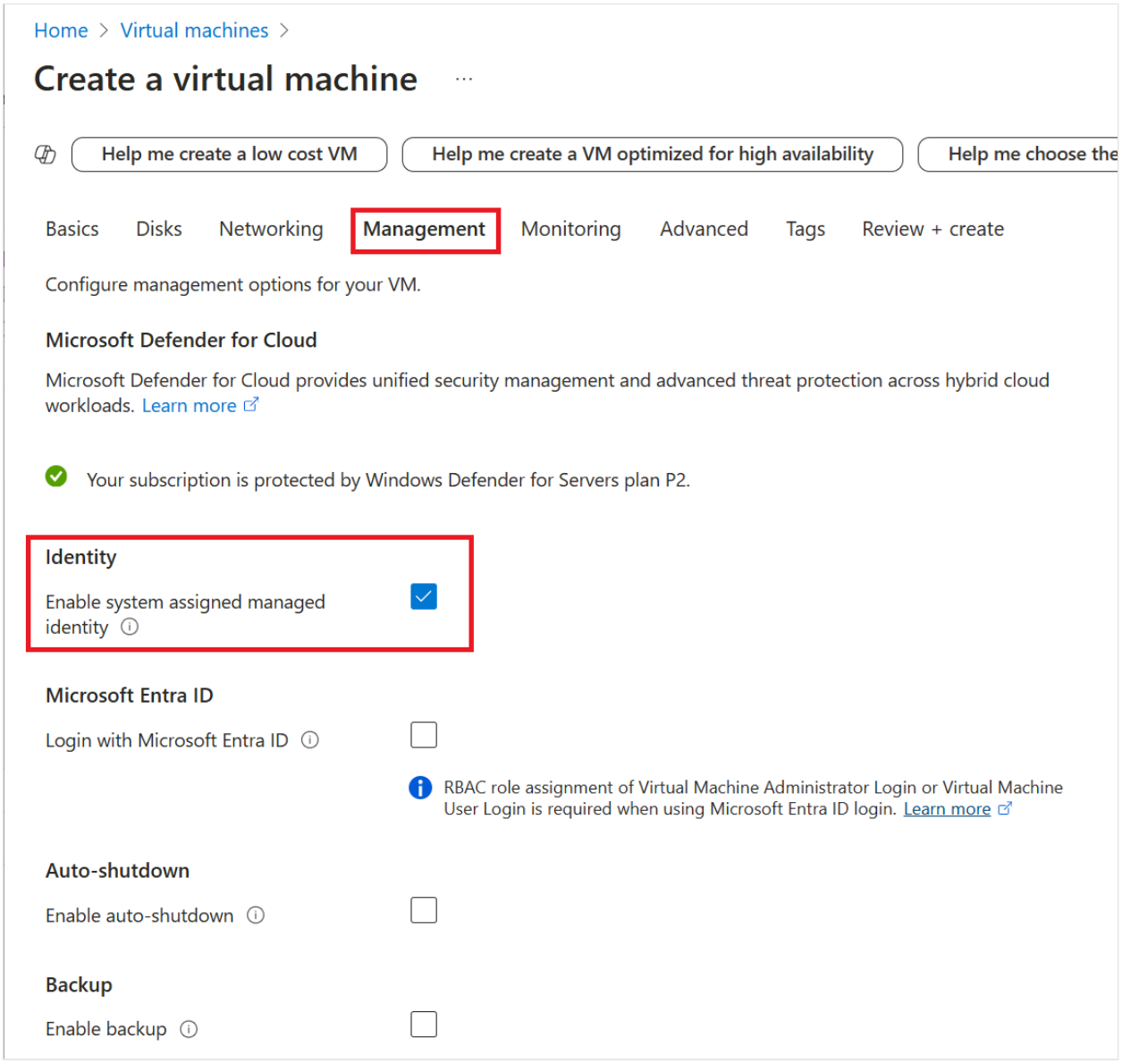
In this section, you learn how to enable and disable the system-assigned managed identity for VM using the Azure portal.

Enable system-assigned managed identity during creation of a VM

To enable system-assigned managed identity on a VM during its creation, your account needs the Virtual Machine Contributor role assignment. No other Microsoft Entra directory role assignments are required.

When creating a Windows virtual machine or Linux virtual machine, select the **Management** tab.

In the **Identity** section, select the **Enable system assigned managed identity** check-box.

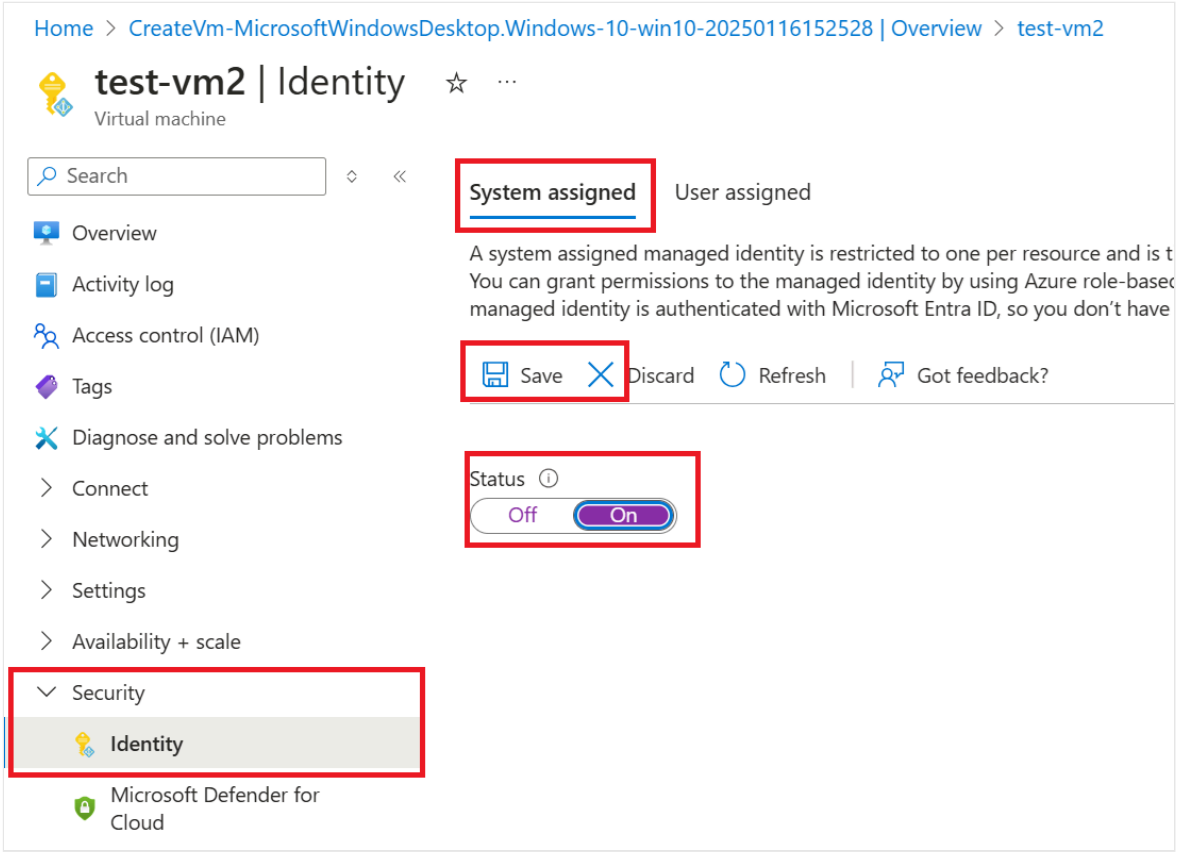


Enable system-assigned managed identity on an existing VM

> **Tip**
>
> Steps in this article might vary slightly based on the portal you start from.

To enable system-assigned managed identity on a VM that was originally provisioned without it, your account needs the Virtual Machine Contributor role assignment. No other Microsoft Entra directory role assignments are required.

1. Sign in to the Azure portal using an account associated with the Azure subscription that contains the VM.
2. Navigate to the desired Virtual Machine and in the **Security** section select **Identity**.
3. Under **System assigned**, **Status**, select **On** and then click **Save**:

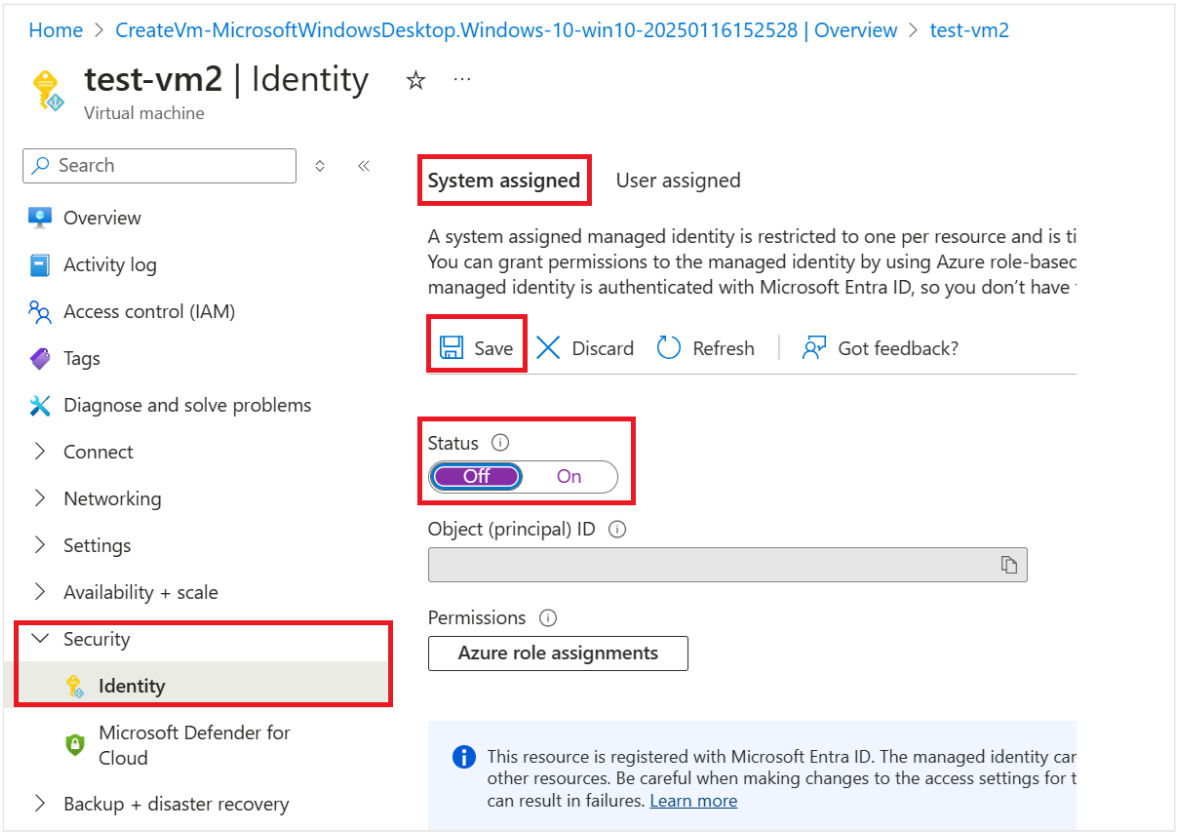


Remove system-assigned managed identity from a VM

To remove system-assigned managed identity from a VM, your account needs the Virtual Machine Contributor role assignment. No other Microsoft Entra directory role assignments are required.

If you have a Virtual Machine that no longer needs system-assigned managed identity:

1. Sign in to the Azure portal using an account associated with the Azure subscription that contains the VM.
2. Navigate to the desired Virtual Machine and in the **Security** section select **Identity**.
3. Under **System assigned**, **Status**, select **Off** and then click **Save**:



User-assigned managed identity

In this section, you learn how to add and remove a user-assigned managed identity from a VM using the Azure portal.

Assign a user-assigned identity during the creation of a VM

To assign a user-assigned identity to a VM, your account needs the Virtual Machine Contributor and Managed Identity Operator role assignments. No other Microsoft Entra directory role assignments are required.

Currently, the Azure portal does not support assigning a user-assigned managed identity during the creation of a VM. First create a Windows virtual machine or Linux virtual machine, then assign a user-assigned managed identity to the VM.

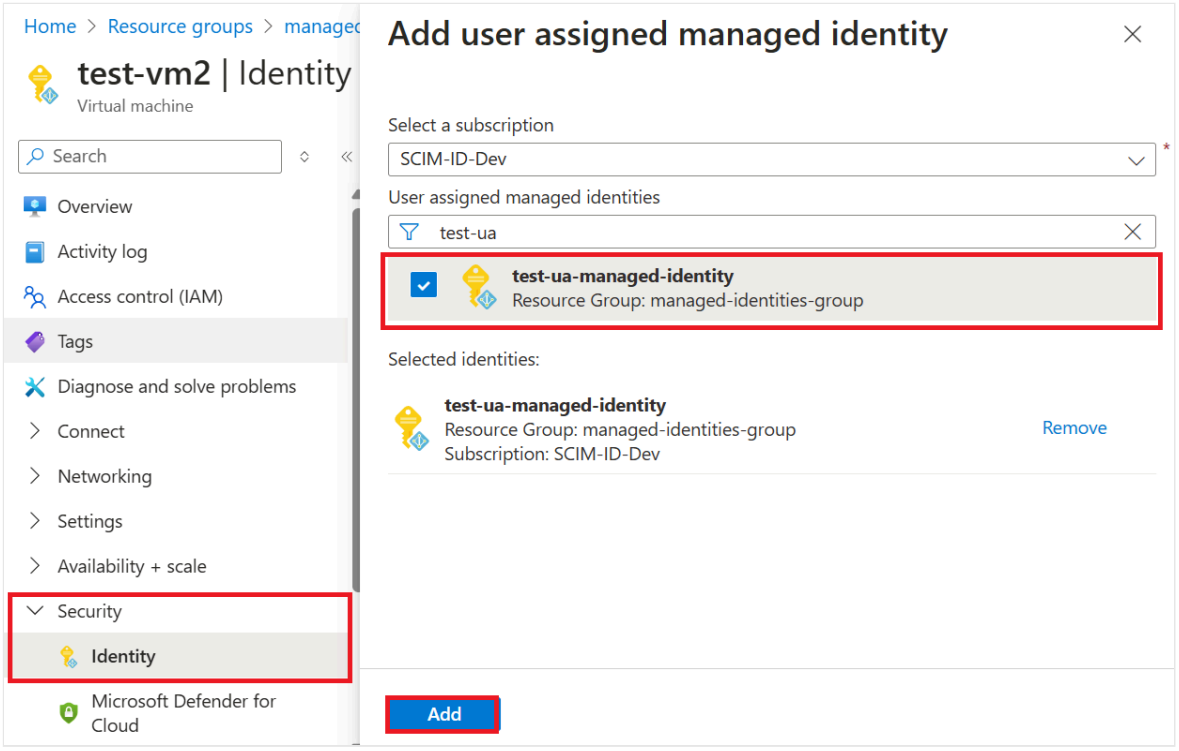
Assign a user-assigned managed identity to an existing VM

To assign a user-assigned identity to a VM, your account needs the Virtual Machine Contributor and Managed Identity Operator role assignments. No other Microsoft Entra directory role assignments are required.

1. Sign in to the Azure portal using an account associated with the Azure subscription that contains the VM.

2. Navigate to the desired VM and click **Security** > **Identity**, **User assigned** and then **+Add**. Click the user-assigned identity you want to add to the VM and then click **Add**.

3. Select the previously created user assigned managed identity from the list.

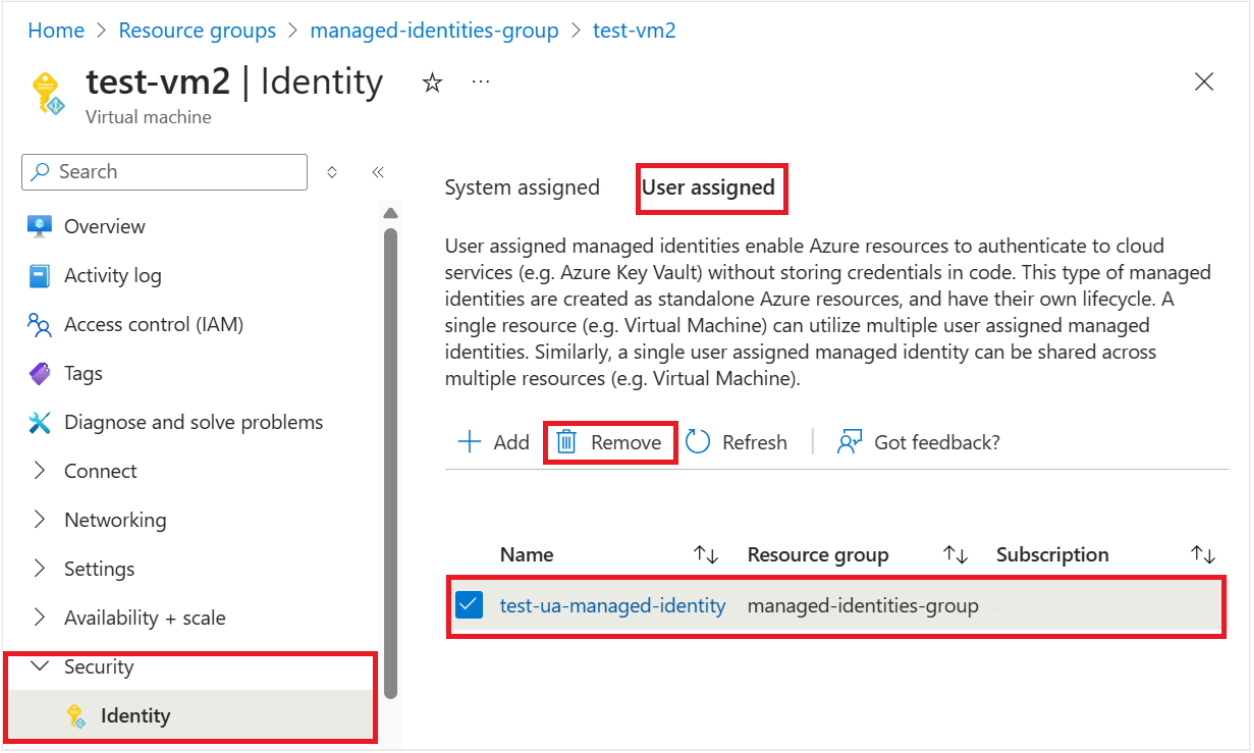


Remove a user-assigned managed identity from a VM

To remove a user-assigned identity from a VM, your account needs the Virtual Machine Contributor role assignment. No other Microsoft Entra directory role assignments are required.

Sign in to the Azure portal using an account associated with the Azure subscription that contains the VM.

Navigate to the desired VM and select **Security** > **Identity**, **User assigned**, the name of the user-assigned managed identity you want to delete and then click **Remove** (click **Yes** in the confirmation pane).



Next steps

- Using the Azure portal, give an Azure VM's managed identity access to another Azure resource.

Feedback

Was this page helpful? Yes No

Provide product feedback

Configure managed identities for Azure resources on a virtual machine scale set

Article • 01/16/2025

Managed identities for Azure resources is a feature of Microsoft Entra ID. Each of the Azure services that support managed identities for Azure resources are subject to their own timeline. Make sure you review the availability status of managed identities for your resource and known issues before you begin.

Managed identities for Azure resources provide Azure services with an automatically managed identity in Microsoft Entra ID. You can use this identity to authenticate to any service that supports Microsoft Entra authentication, without having credentials in your code.

For information about Azure Policy definition and details, see Use Azure Policy to assign managed identities (preview).

In this article, using the Azure portal, you learn how to perform the following managed identities for Azure resources operations on a virtual machine scale set:

- If you're unfamiliar with managed identities for Azure resources, check out the overview section.
- If you don't already have an Azure account, sign up for a free account before continuing.
- To perform the management operations in this article, your account needs the following Azure role assignments:

  > **Note**
  >
  > No additional Microsoft Entra directory role assignments required.

  - Virtual Machine Contributor to enable and remove system-assigned managed identity from a virtual machine scale set.

System-assigned managed identity

In this section, you will learn how to enable and disable the system-assigned managed identity using the Azure portal.

Enable system-assigned managed identity during creation of a virtual machine scale set

Currently, the Azure portal does not support enabling system-assigned managed identity during the creation of a virtual machine scale set. First create a Virtual Machine Scale Set, then enable a system-assigned managed identity on the scale set:

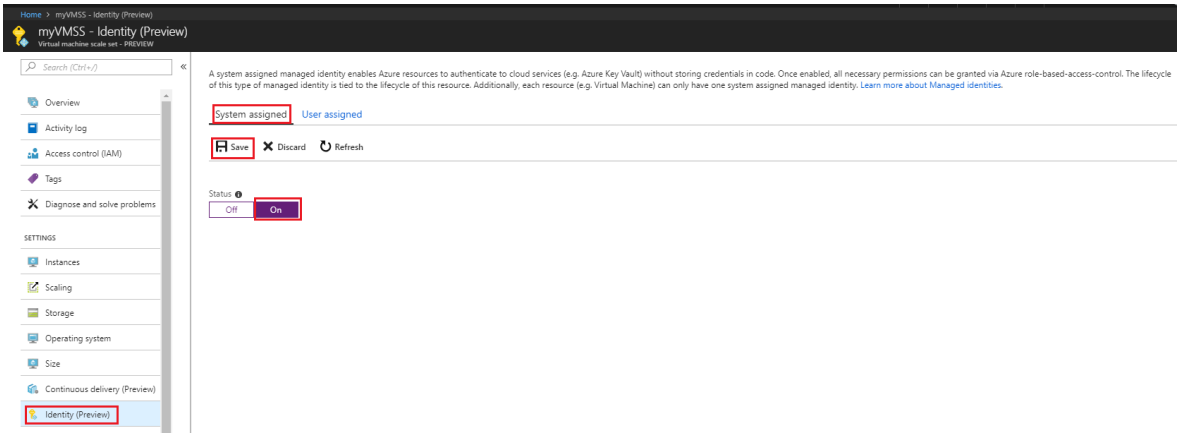
- Create a Virtual Machine Scale Set in the Azure portal

Enable system-assigned managed identity on an existing virtual machine scale set

> **Tip**
>
> Steps in this article might vary slightly based on the portal you start from.

To enable the system-assigned managed identity on a virtual machine scale set that was originally provisioned without it:

1. Sign in to the Azure portal using an account associated with the Azure subscription that contains the virtual machine scale set.
2. Navigate to the desired virtual machine scale set.
3. Under **System assigned**, **Status**, select **On** and then click **Save**:



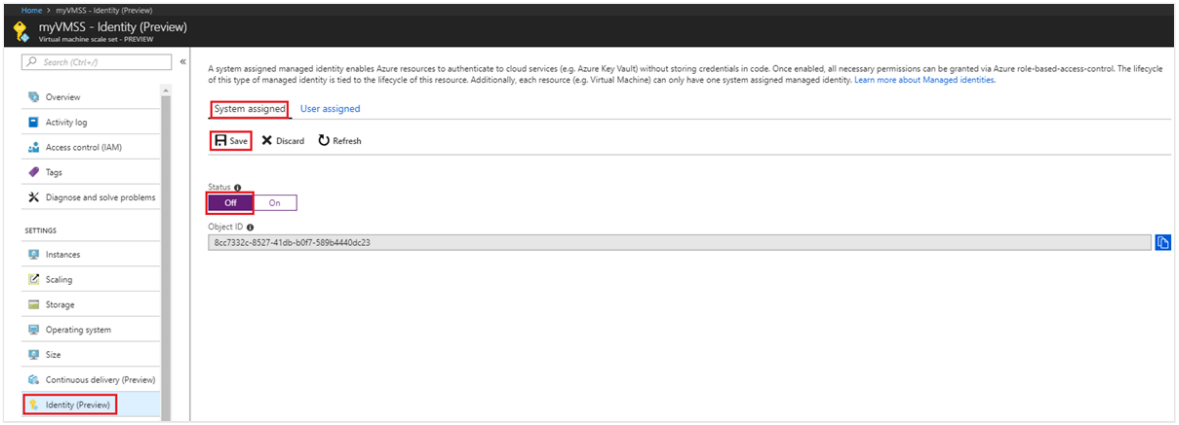
Remove system-assigned managed identity from a virtual machine scale set

If you have a virtual machine scale set that no longer needs a system-assigned managed identity:

1. Sign in to the Azure portal using an account associated with the Azure subscription that contains the virtual machine scale set. Also make sure your account belongs to a role that gives you write permissions on the virtual machine scale set.

2. Navigate to the desired virtual machine scale set.

3. Under **System assigned**, **Status**, select **Off** and then click **Save**:



User-assigned managed identity

In this section, you learn how to add and remove a user-assigned managed identity from a virtual machine scale set using the Azure portal.

Assign a user-assigned managed identity during the creation of a virtual machine scale set

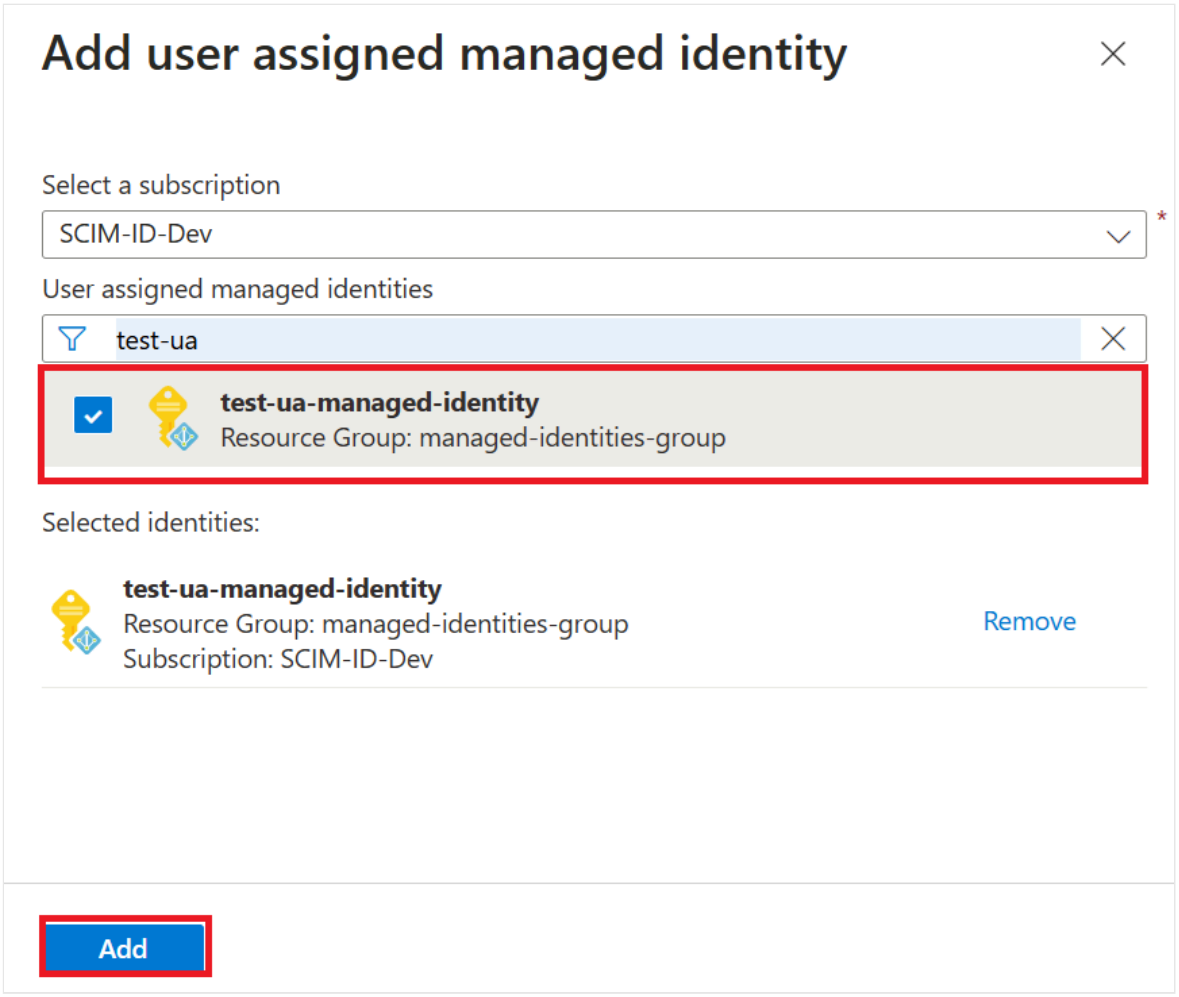
Currently, the Azure portal does not support assigning a user-assigned managed identity during the creation of a virtual machine scale set. First create a Virtual Machine Scale Set, then assign a user assigned managed identity to the scale set.

Assign a user-assigned managed identity to an existing virtual machine scale set

1. Sign in to the Azure portal using an account associated with the Azure subscription that contains the virtual machine scale set.

2. Navigate to the desired virtual machine scale set and click **Security** > **Identity**, **User assigned** and then **+Add**.

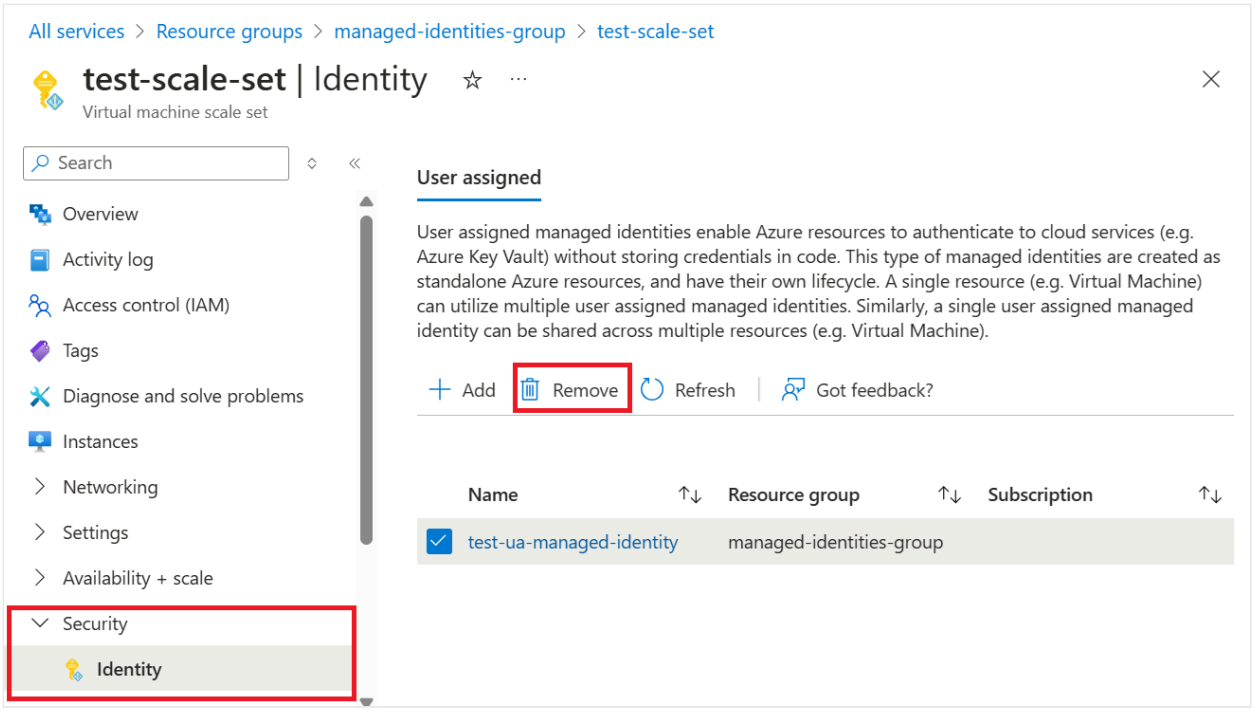
3. Select the previously created user assigned managed identity from the list.



Remove a user-assigned managed identity from a virtual machine scale set

Sign in to the Azure portal using an account associated with the Azure subscription that contains the VM.

Navigate to the desired virtual machine scale set and click **Identity**, **User assigned**, the name of the user-assigned managed identity you want to delete and then click **Remove** (click **Yes** in the confirmation pane).



Next steps

- Using the Azure portal, give an Azure virtual machine scale set managed identity access to another Azure resource.

Feedback

Was this page helpful? Yes No

Provide product feedback

Use Azure Policy to assign managed identities (preview)

Article • 09/05/2024

Azure Policy helps enforce organizational standards and assess compliance at scale. Through its compliance dashboard, Azure policy provides an aggregated view that helps administrators evaluate the overall state of the environment. You have the ability to drill down to the per-resource, per-policy granularity. It also helps bring your resources to compliance through bulk remediation for existing resources and automatic remediation for new resources. Common use cases for Azure Policy include implementing governance for:

- Resource consistency
- Regulatory compliance
- Security
- Cost
- Management

Policy definitions for these common use cases are already available in your Azure environment to help you get started.

Azure Monitoring Agents require a managed identity on the monitored Azure Virtual Machines (VMs). This document describes the behavior of a built-in Azure Policy provided by Microsoft that helps ensure a managed identity, needed for these scenarios, is assigned to VMs at scale.

While using system-assigned managed identity is possible, when used at scale (for example, for all VMs in a subscription) it results in substantial number of identities created (and deleted) in Microsoft Entra ID. To avoid this churn of identities, it is recommended to use user-assigned managed identities, which can be created once and shared across multiple VMs.

Policy definition and details

- Policy for Virtual Machines
- Policy for Virtual Machine Scale Sets

When executed, the policy takes the following actions:

1. Create, if not exist, a new built-in user-assigned managed identity in the subscription and each Azure region based on the VMs that are in scope of the

policy.
2. Once created, put a lock on the user-assigned managed identity so that it will not be accidentally deleted.
3. Assign the built-in user-assigned managed identity to Virtual Machines from the subscription and region based on the VMs that are in scope of the policy.

> ⓘ **Note**
>
> If the Virtual Machine has exactly 1 user-assigned managed identity already assigned, then the policy skips this VM to assign the built-in identity. This is to make sure assignment of the policy does not break applications that take a dependency on **the default behavior of the token endpoint on IMDS.**

There are two scenarios to use the policy:

- Let the policy create and use a “built-in” user-assigned managed identity.
- Bring your own user-assigned managed identity.

The policy takes the following input parameters:

- Bring-Your-Own-UAMI? - Should the policy create, if not exist, a new user-assigned managed identity?
  - If set to true, then you must specify:
    - Name of the managed identity.
    - Resource group containing the managed identity.
  - If set to false, then no additional input is needed.
    - The policy will create the required user-assigned managed identity called “built-in-identity” in a resource group called “built-in-identity-rg".
- Restrict-Bring-Your-Own-UAMI-To-Subscription? - When the Bring-Your-Own-UAMI parameter is set to true, should the policy utilize a centralized user-assigned managed identity or utilize an identity for each subscription?
  - If set to true, then no additional input is needed.
    - The policy will use a user-assigned managed identity per subscription.
  - If set to false, the policy will utilize a single centralized user assigned managed identity that will be applied across all the subscriptions covered by the policy assignment. You must specify:
    - User Assigned Managed Identity Resource Id

Using the policy

Creating the policy assignment

The policy definition can be assigned to different scopes in Azure – at the management group subscription or a specific resource group. As policies need to be enforced all the time, the assignment operation is performed using a managed identity associated with the policy-assignment object. The policy assignment object supports both system-assigned and user-assigned managed identity. For example, Joe can create a user-assigned managed identity called PolicyAssignmentMI. The built-in policy creates a user-assigned managed identity in each subscription and in each region with resources that are in scope of the policy assignment. The user-assigned managed identities created by the policy has the following resourceId format:

> /subscriptions/your-subscription-id/resourceGroups/built-in-identity-rg/providers/Microsoft.ManagedIdentity/userAssignedIdentities/built-in-identity-{location}

For example:

> /subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-eeeeee4e4e4e/resourceGroups/built-in-identity-rg/providers/Microsoft.ManagedIdentity/userAssignedIdentities/built-in-identity-eastus

Required authorization

For PolicyAssignmentMI managed identity to be able to assign the built-in policy across the specified scope, it needs the following permissions, expressed as an Azure RBAC (Azure role-based access control) Role Assignment:

Expand table

Principal	Role / Action	Scope	Purpose
PolicyAssigmentMI	Managed Identity Operator	/subscription/subscription-id/resourceGroups/built-in-identity OR Bring-your-own-User-assinged-Managed identity	Required to assign the built-in identity to VMs.
PolicyAssigmentMI	Contributor	/subscription/subscription-id>	Required to create the resource-group that holds the built-in

Principal	Role / Action	Scope	Purpose
			managed identity in the subscription.
PolicyAssigmentMI	Managed Identity Contributor	/subscription/subscription-id/resourceGroups/built-in-identity	Required to create a new user-assigned managed identity.
PolicyAssigmentMI	User Access Administrator	/subscription/subscription-id/resourceGroups/built-in-identity OR Bring-your-own-User-assigned-Managed identity	Required to set a lock on the user-assigned managed identity created by the policy.

As the policy assignment object must have this permission ahead of time, PolicyAssignmentMI cannot be a system-assigned managed identity for this scenario. The user performing the policy assignment task must pre-authorize PolicyAssignmentMI ahead of time with the above role assignments.

As you can see the resultant least privilege role required is “contributor” at the subscription scope.

Known issues

Possible race condition with another deployment that changes the identities assigned to a VM can result in unexpected results.

If there are two or more parallel deployments updating the same virtual machine and they all change the identity configuration of the virtual machine, then it is possible, under specific race conditions, that all expected identities will NOT be assigned to the machines. For example, if the policy in this document is updating the managed identities of a VM and at the same time another process is also making changes to the managed identities section, then it is not guaranteed that all the expected identities are properly assigned to the VM.

Next steps

- Deploy Azure Monitor Agent

Feedback

Was this page helpful? Yes No

Provide product feedback

Configure an application to trust a managed identity (preview)

Article • 04/10/2025

This article describes how to configure a Microsoft Entra application to trust a managed identity. You can then exchange the managed identity token for an access token that can access Microsoft Entra protected resources without needing to use or manage App secrets.

Prerequisites

- An Azure account with an active subscription. Create an account for free.
- This Azure account must have permissions to update application credentials. Any of the following Microsoft Entra roles include the required permissions:
  - Application Administrator
  - Application Developer
  - Cloud Application Administrator
- An understanding of the concepts in managed identities for Azure resources.
- A user-assigned managed identity assigned to the Azure compute resource (for example, a virtual machine or Azure App Service) that hosts your workload.
- An app registration in Microsoft Entra ID. This app registration must belong to the same tenant as the managed identity
  - If you need to access resources in another tenant, your app registration must be a multitenant application and provisioned into the other tenant. Learn about how to add a multitenant app in other tenants.
- The app registration must have access granted to Microsoft Entra protected resources (for example, Azure, Microsoft Graph, Microsoft 365, etc.). This access can be granted through API permissions or delegated permissions.

Important considerations and restrictions

To create, update, or delete a federated identity credential, the account performing the action must have the Application Administrator, Application Developer, Cloud Application Administrator, or Application Owner role. The microsoft.directory/applications/credentials/update permission is required to update a federated identity credential.

A maximum of 20 federated identity credentials can be added to an application or user-assigned managed identity.

When you configure a federated identity credential, there are several important pieces of information to provide:

- *issuer*, *subject* are the key pieces of information needed to set up the trust relationship. When the Azure workload requests Microsoft identity platform to exchange the managed identity token for an Entra app access token, the *issuer* and *subject* values of the federated identity credential are checked against the `issuer` and `subject` claims provided in the Managed Identity token. If that validation check passes, Microsoft identity platform issues an access token to the external software workload.

- *issuer* is the URL of the Microsoft Entra tenant's Authority URL in the form `https://login.microsoftonline.com/{tenant}/v2.0`. Both the Microsoft Entra app and managed identity must belong to the same tenant. If the `issuer` claim has leading or trailing whitespace in the value, the token exchange is blocked.

- `subject`: This is the case-sensitive GUID of the managed identity's **Object (Principal) ID** assigned to the Azure workload. The managed identity must be in the same tenant as the app registration, even if the target resource is in a different cloud. The Microsoft identity platform will reject the token exchange if the `subject` in the federated identity credential configuration does not exactly match the managed identity's Principal ID.

  > **ⓘ Important**
  >
  > Only user-assigned managed identities can be used as a federated credential for apps. system-assigned identities aren't supported.

- *audiences* specifies the value that appears in the `aud` claim in the managed identity token (Required). The value must be one of the following depending on the target cloud.
  - **Microsoft Entra ID global service**: `api://AzureADTokenExchange`
  - **Microsoft Entra ID for US Government**: `api://AzureADTokenExchangeUSGov`
  - **Microsoft Entra China operated by 21Vianet**: `api://AzureADTokenExchangeChina`

  > **ⓘ Important**
  >
  > Accessing resources in *another tenant* is supported. Accessing resources in *another cloud* is not supported. Token requests to other clouds will fail.

  > **ⓘ Important**

If you accidentally add incorrect information in the *issuer*, *subject* or *audience* setting the federated identity credential is created successfully without error. The error does not become apparent until the token exchange fails.

- *name* is the unique identifier for the federated identity credential. (Required) This field has a character limit of 3-120 characters and must be URL friendly. Alphanumeric, dash, or underscore characters are supported, and the first character must be alphanumeric only. It's immutable once created.
- *description* is the user-provided description of the federated identity credential (Optional). The description isn't validated or checked by Microsoft Entra ID. This field has a limit of 600 characters.

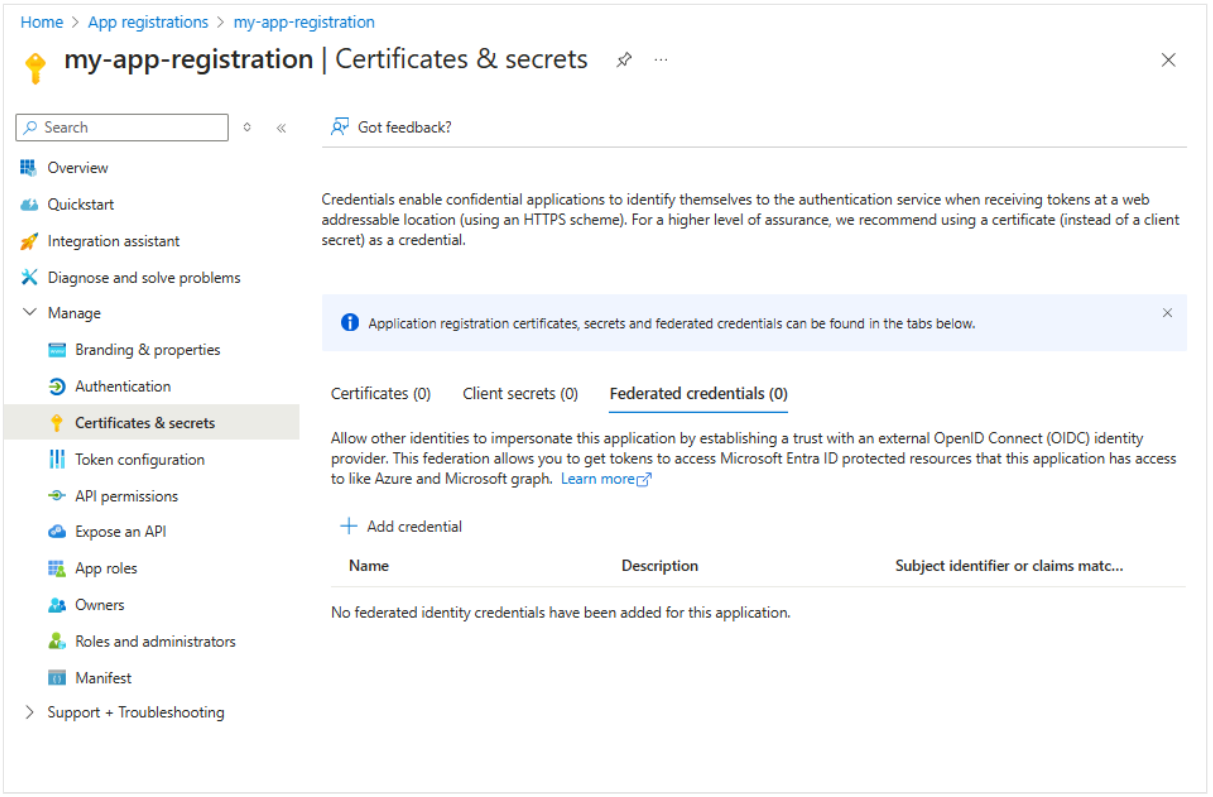
Wildcard characters aren't supported in any federated identity credential property value.

Configure a federated identity credential on an application

In this section, you'll configure a federated identity credential on an existing application to trust a managed identity. Use the following tabs to choose how to configure a federated identity credential on an existing application.

Microsoft Entra admin center

1. Sign in to the Microsoft Entra admin center. Check that you are in the tenant where your application is registered.
2. Browse to **Identity** > **Applications** > **App registrations**, and select your application in the main window.
3. Under **Manage**, select **Certificates & secrets**.
4. Select the Federated credentials tab and select **Add credential**.

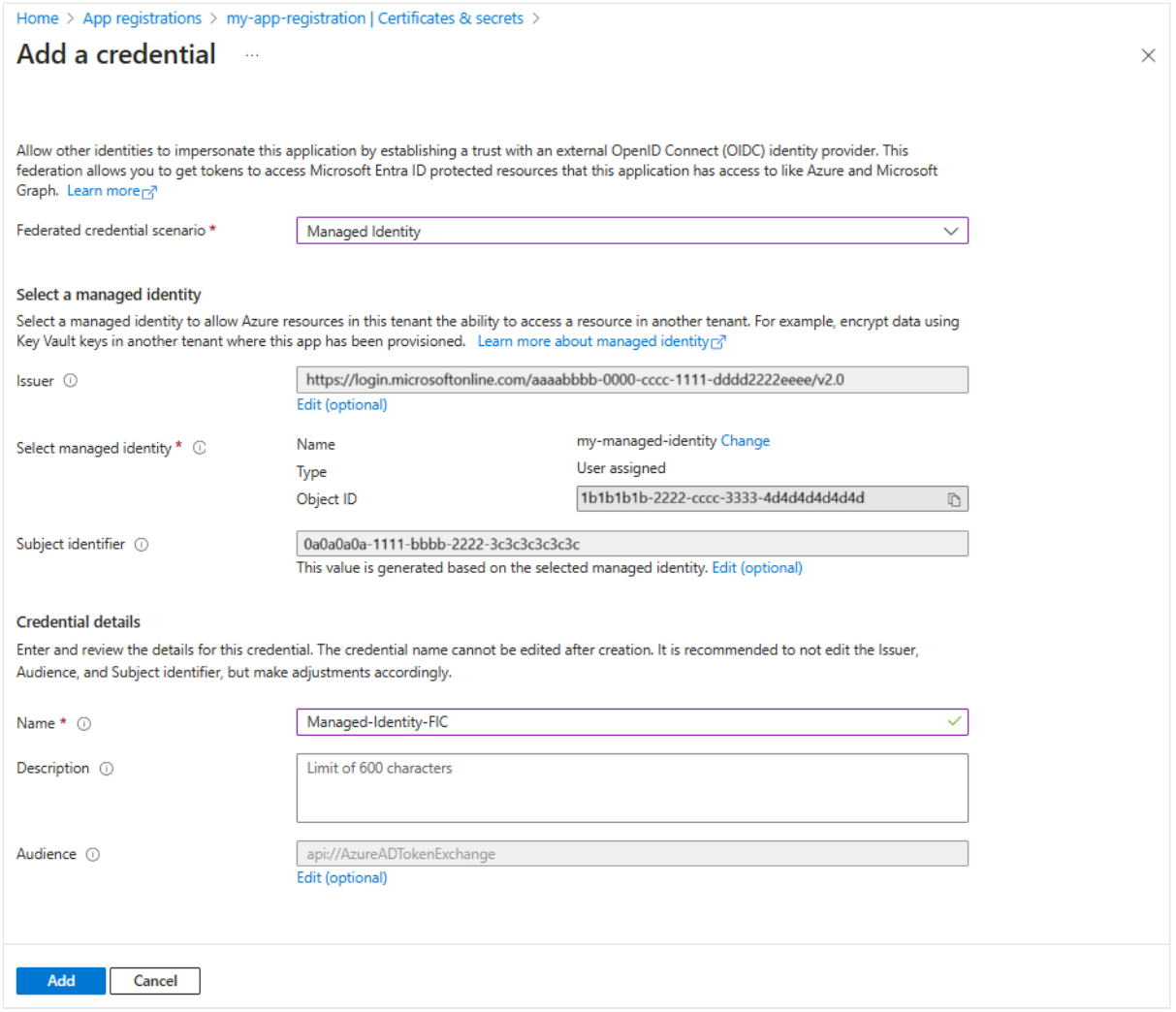


5. From the **Federated credential scenario** dropdown, select **Managed Identity** and fill in the values according to the following table:

Expand table

Field	Description	Example
Issuer	The OAuth 2.0 / OIDC issuer URL of the Microsoft Entra ID authority that issues the managed identity token. This value is automatically populated with the current Entra tenant issuer.	`https://login.microsoftonline.com/{tenantID}/v2.0`
Select managed identity	Click on this link to select the managed identity that will act as the federated identity credential. You can only use User-Assigned Managed Identities as a credential.	*msi-webapp1*

Field	Description	Example
Description (Optional)	A user-provided description of the federated identity credential.	*Trust the workloads UAMI as a credential to my App*
Audience	The audience value that must appear in the external token.	Must be set to one of the following values: • **Entra ID Global Service**: *api://AzureADTokenExchange* • **Entra ID for US Government**: *api://AzureADTokenExchangeUSGov* • **Entra ID China operated by 21Vianet**: *api://AzureADTokenExchangeChina*



Update your application code to request an access token

The following code snippets demonstrate how to acquire a managed identity token and use it as a credential for your Entra application. The samples are valid in both cases where the target resource in the same tenant as the Entra application, or in a different tenant.

Azure.Identity

This example demonstrates accessing an Azure storage container, but can be adapted to access any resource protected by Microsoft Entra.

C#

```
using Azure.Identity;
using Azure.Storage.Blobs;

internal class Program
{
  // This example demonstrates how to access an Azure blob storage account by
utilizing the manage identity credential.
  static void Main(string[] args)
  {
    string storageAccountName = "YOUR_STORAGE_ACCOUNT_NAME";
    string containerName = "CONTAINER_NAME";

    // The application must be granted access on the target resource
    string appClientId = "YOUR_APP_CLIENT_ID";

    // The tenant where the target resource is created, in this example, the
storage account tenant
    // If the resource tenant different from the app tenant, your app needs to be
    string resourceTenantId = "YOUR_RESOURCE_TENANT_ID";

    // The managed identity which you configured as a Federated Identity
Credential (FIC)
    string miClientId = "YOUR_MANAGED_IDENTITY_CLIENT_ID";

    // Audience value must be one of the below values depending on the target
cloud.
    // Entra ID Global cloud: api://AzureADTokenExchange
    // Entra ID US Government: api://AzureADTokenExchangeUSGov
    // Entra ID China operated by 21Vianet: api://AzureADTokenExchangeChina
    string audience = "api://AzureADTokenExchange";

    // 1. Create an assertion with the managed identity access token, so that it
can be exchanged an app token
    var miCredential = new ManagedIdentityCredential(managedIdentityClientId);
    ClientAssertionCredential assertion = new(
        tenantId,
        appClientId,
        async (token) =>
        {
            // fetch Managed Identity token for the specified audience
```

```
            var tokenRequestContext = new Azure.Core.TokenRequestContext(new[] {
$"{audience}/.default" });
            var accessToken = await
miCredential.GetTokenAsync(tokenRequestContext).ConfigureAwait(false);
            return accessToken.Token;
        });

        // 2. The assertion can be used to obtain an App token (taken care of by
the SDK)
        var containerClient  = new BlobContainerClient(new
Uri($"https://{storageAccountName}.blob.core.windows.net/{containerName}"),
assertion);

        await foreach (BlobItem blob in containerClient.GetBlobsAsync())
        {
            // TODO: perform operations with the blobs
            BlobClient blobClient = containerClient.GetBlobClient(blob.Name);
            Console.WriteLine($"Blob name: {blobClient.Name}, uri:
{blobClient.Uri}");
        }
    }
}
```

Microsoft.Identity.Web

In **Microsoft.Identity.Web**, you can set the `ClientCredentials` section in your *appsettings.json* to use `SignedAssertionFromManagedIdentity` to enable your code use the configured managed identity as a credential:

JSON

```
{
  "AzureAd": {
    "Instance": "https://login.microsoftonline.com/",
    "ClientId": "YOUR_APPLICATION_ID",
    "TenantId": "YOUR_TENANT_ID",

    "ClientCredentials": [
      {
        "SourceType": "SignedAssertionFromManagedIdentity",
        "ManagedIdentityClientId":
"YOUR_USER_ASSIGNED_MANAGED_IDENTITY_CLIENT_ID",
        "TokenExchangeUrl": "api://AzureADTokenExchange"
      }
    ]
  }
}
```

MSAL (.NET)

In **MSAL**, you can use the ManagedClientApplication class to acquire a Managed Identity token. This token can then be used as a client assertion when constructing a confidential client application.

C#

```
using Microsoft.Identity.Client;
using Azure.Storage.Blobs;
using Azure.Core;

internal class Program
{
  static async Task Main(string[] args)
  {
      string storageAccountName = "YOUR_STORAGE_ACCOUNT_NAME";
      string containerName = "CONTAINER_NAME";

      string appClientId = "YOUR_APP_CLIENT_ID";
      string resourceTenantId = "YOUR_RESOURCE_TENANT_ID";
      Uri authorityUri =
new($"https://login.microsoftonline.com/{resourceTenantId}");
      string miClientId = "YOUR_MI_CLIENT_ID";
      string audience = "api://AzureADTokenExchange";

      // Get mi token to use as assertion
      var miAssertionProvider = async (AssertionRequestOptions _) =>
      {
            var miApplication = ManagedIdentityApplicationBuilder
                .Create(ManagedIdentityId.WithUserAssignedClientId(miClientId))
                .Build();

            var miResult = await
miApplication.AcquireTokenForManagedIdentity(audience)
                .ExecuteAsync()
                .ConfigureAwait(false);
            return miResult.AccessToken;
      };

      // Create a confidential client application with the assertion.
      IConfidentialClientApplication app =
ConfidentialClientApplicationBuilder.Create(appClientId)
        .WithAuthority(authorityUri, false)
        .WithClientAssertion(miAssertionProvider)
        .WithCacheOptions(CacheOptions.EnableSharedCacheOptions)
        .Build();

        // Get the federated app token for the storage account
        string[] scopes =
[$"https://{storageAccountName}.blob.core.windows.net/.default"];
        AuthenticationResult result = await
app.AcquireTokenForClient(scopes).ExecuteAsync().ConfigureAwait(false);

        TokenCredential tokenCredential = new
```

```
AccessTokenCredential(result.AccessToken);
        var client = new BlobContainerClient(
            new
Uri($"https://{storageAccountName}.blob.core.windows.net/{containerName}"),
            tokenCredential);

        await foreach (BlobItem blob in containerClient.GetBlobsAsync())
        {
            // TODO: perform operations with the blobs
            BlobClient blobClient = containerClient.GetBlobClient(blob.Name);
            Console.WriteLine($"Blob name: {blobClient.Name}, URI:
{blobClient.Uri}");
        }
    }
}
```

See also

- Important considerations and restrictions for federated identity credentials.

How to use managed identities for Azure resources on an Azure VM to acquire an access token

Article • 02/27/2025

Managed identities for Azure resources provide Azure services with an automatically managed identity in Microsoft Entra ID. You can use this identity to authenticate to any service that supports Microsoft Entra authentication, without having credentials in your code.

This article provides various code and script examples for token acquisition. It also contains guidance about handling token expiration and HTTP errors.

Prerequisites

- If you're not familiar with the managed identities for Azure resources feature, see this overview. If you don't have an Azure account, sign up for a free account before you continue.

If you plan to use the Azure PowerShell examples in this article, be sure to install the latest version of Azure PowerShell.

> **Important**
>
> - All sample code/script in this article assumes the client is running on a virtual machine with managed identities for Azure resources. Use the virtual machine "Connect" feature in the Azure portal, to remotely connect to your VM. For details on enabling managed identities for Azure resources on a VM, see Configure managed identities for Azure resources on a VM using the Azure portal, or one of the variant articles (using PowerShell, CLI, a template, or an Azure SDK).

> **Important**
>
> - The security boundary of managed identities for Azure resources is the resource where the identity is used. All code/scripts running on a virtual

machine can request and retrieve tokens for any managed identities available on it.

Overview

A client application can request a managed identity app-only access token to access a given resource. The token is based on the managed identities for Azure resources service principal. As such, there's no need for the client to obtain an access token under its own service principal. The token is suitable for use as a bearer token in service-to-service calls requiring client credentials.

Expand table

Link	Description
Get a token using HTTP	Protocol details for managed identities for Azure resources token endpoint
Get a token using Azure.Identity	Get a token using Azure.Identity library
Get a token using the Microsoft.Azure.Services.AppAuthentication library for .NET	Example of using the Microsoft.Azure.Services.AppAuthentication library from a .NET client
Get a token using C#	Example of using managed identities for Azure resources REST endpoint from a C# client
Get a token using Java	Example of using managed identities for Azure resources REST endpoint from a Java client
Get a token using Go	Example of using managed identities for Azure resources REST endpoint from a Go client
Get a token using PowerShell	Example of using managed identities for Azure resources REST endpoint from a PowerShell client
Get a token using CURL	Example of using managed identities for Azure resources REST endpoint from a Bash/CURL client
Handling token caching	Guidance for handling expired access tokens
Error handling	Guidance for handling HTTP errors returned from the managed identities for Azure resources token endpoint

Link	Description
Resource IDs for Azure services	Where to get resource IDs for supported Azure services

Get a token using HTTP

The fundamental interface for acquiring an access token is based on REST, making it accessible to any client application running on the VM that can make HTTP REST calls. This approach is similar to the Microsoft Entra programming model, except the client uses an endpoint on the virtual machine (vs a Microsoft Entra endpoint).

Sample request using the Azure Instance Metadata Service (IMDS) endpoint *(recommended)*:

```
GET 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https://management.azure.com/' HTTP/1.1 Metadata: true
```

Expand table

Element	Description
`GET`	The HTTP verb, indicating you want to retrieve data from the endpoint. In this case, an OAuth access token.
`http://169.254.169.254/metadata/identity/oauth2/token`	The managed identities for Azure resources endpoint for the Instance Metadata Service.
`api-version`	A query string parameter, indicating the API version for the IMDS endpoint. Use API version `2018-02-01` or greater.
`resource`	A query string parameter, indicating the App ID URI of the target resource. It also appears in the `aud` (audience) claim of the issued token. This example requests a token to access Azure Resource Manager, which has an App ID URI of `https://management.azure.com/`.

Element	Description
`Metadata`	An HTTP request header field required by managed identities. This information is used as a mitigation against server side request forgery (SSRF) attacks. This value must be set to "true", in all lower case.
`object_id`	(Optional) A query string parameter, indicating the object_id of the managed identity you would like the token for. Required, if your VM has multiple user-assigned managed identities.
`client_id`	(Optional) A query string parameter, indicating the client_id of the managed identity you would like the token for. Required, if your VM has multiple user-assigned managed identities.
`msi_res_id`	(Optional) A query string parameter, indicating the msi_res_id (Azure Resource ID) of the managed identity you would like the token for. Required, if your VM has multiple user-assigned managed identities.

Sample response:

JSON

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "access_token": "eyJ0eXAi...",
  "refresh_token": "",
  "expires_in": "3599",
  "expires_on": "1506484173",
  "not_before": "1506480273",
  "resource": "https://management.azure.com/",
  "token_type": "Bearer"
}
```

Expand table

Element	Description
`access_token`	The requested access token. When you call a secured REST API, the token is embedded in the `Authorization` request header field as a "bearer" token, allowing the API to authenticate the caller.
`refresh_token`	Not used by managed identities for Azure resources.
`expires_in`	The number of seconds the access token continues to be valid, before expiring, from time of issuance. Time of issuance can be found in the token's `iat` claim.
`expires_on`	The timespan when the access token expires. The date is represented as the number of seconds from "1970-01-01T0:0:0Z UTC" (corresponds to the token's `exp` claim).
`not_before`	The timespan when the access token takes effect, and can be accepted. The date is represented as the number of seconds from "1970-01-01T0:0:0Z UTC" (corresponds to the token's `nbf` claim).
`resource`	The resource the access token was requested for, which matches the `resource` query string parameter of the request.
`token_type`	The type of token, which is a "Bearer" access token, which means the resource can give access to the bearer of this token.

Get a token using the Azure identity client library

Using the Azure identity client library is the recommended way to use managed identities. All Azure SDKs are integrated with the `Azure.Identity` library that provides support for DefaultAzureCredential. This class makes it easy to use Managed Identities with Azure SDKs.Learn more

1. Install the Azure.Identity package and other required Azure SDK library packages, such as Azure.Security.KeyVault.Secrets.

2. Use the sample code below. You don't need to worry about getting tokens. You can directly use the Azure SDK clients. The code is for demonstrating how to get the token, if you need to.

```
C#

using Azure.Core;
using Azure.Identity;

string userAssignedClientId = "<your managed identity client Id>";
var credential = new DefaultAzureCredential(new
```

```
DefaultAzureCredentialOptions { ManagedIdentityClientId =
userAssignedClientId });
var accessToken = credential.GetToken(new TokenRequestContext(new[] {
"https://vault.azure.net" }));
// To print the token, you can convert it to string
String accessTokenString = accessToken.Token.ToString();

//You can use the credential object directly with Key Vault client.
var client = new SecretClient(new
Uri("https://myvault.vault.azure.net/"), credential);
```

Get a token using the Microsoft.Azure.Services.AppAuthentication library for .NET

For .NET applications and functions, the simplest way to work with managed identities for Azure resources is through the Microsoft.Azure.Services.AppAuthentication package. This library will also allow you to test your code locally on your development machine. You can test your code using your user account from Visual Studio, the Azure CLI, or Active Directory Integrated Authentication. For more on local development options with this library, see the Microsoft.Azure.Services.AppAuthentication reference. This section shows you how to get started with the library in your code.

1. Add references to the Microsoft.Azure.Services.AppAuthentication and Microsoft.Azure.KeyVault NuGet packages to your application.

2. Add the following code to your application:

```
C#

using Microsoft.Azure.Services.AppAuthentication;
using Microsoft.Azure.KeyVault;
// ...
var azureServiceTokenProvider = new AzureServiceTokenProvider();
string accessToken = await
azureServiceTokenProvider.GetAccessTokenAsync("https://management.azure
.com/");
// OR
var kv = new KeyVaultClient(new
KeyVaultClient.AuthenticationCallback(azureServiceTokenProvider.KeyVaul
tTokenCallback));
```

To learn more about Microsoft.Azure.Services.AppAuthentication and the operations it exposes, see the Microsoft.Azure.Services.AppAuthentication reference and the App Service and KeyVault with managed identities for Azure resources .NET sample.

Get a token using C#

C#

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Web.Script.Serialization;

// Build request to acquire managed identities for Azure resources token
HttpWebRequest request =
(HttpWebRequest)WebRequest.Create("http://169.254.169.254/metadata/identity/
oauth2/token?api-version=2018-02-
01&resource=https://management.azure.com/");
request.Headers["Metadata"] = "true";
request.Method = "GET";

try
{
    // Call /token endpoint
    HttpWebResponse response = (HttpWebResponse)request.GetResponse();

    // Pipe response Stream to a StreamReader, and extract access token
    StreamReader streamResponse = new
StreamReader(response.GetResponseStream());
    string stringResponse = streamResponse.ReadToEnd();
    JavaScriptSerializer j = new JavaScriptSerializer();
    Dictionary<string, string> list = (Dictionary<string, string>)
j.Deserialize(stringResponse, typeof(Dictionary<string, string>));
    string accessToken = list["access_token"];
}
catch (Exception e)
{
    string errorText = String.Format("{0} \n\n{1}", e.Message,
e.InnerException != null ? e.InnerException.Message : "Acquire token
failed");
}
```

Get a token using Java

Use this JSON library to retrieve a token using Java.

Java

```
import java.io.*;
import java.net.*;
import com.fasterxml.jackson.core.*;
```

```
class GetMSIToken {
    public static void main(String[] args) throws Exception {

        URL msiEndpoint = new
URL("http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-
02-01&resource=https://management.azure.com/");
        HttpURLConnection con = (HttpURLConnection)
msiEndpoint.openConnection();
        con.setRequestMethod("GET");
        con.setRequestProperty("Metadata", "true");

        if (con.getResponseCode()!=200) {
            throw new Exception("Error calling managed identity token
endpoint.");
        }

        InputStream responseStream = con.getInputStream();

        JsonFactory factory = new JsonFactory();
        JsonParser parser = factory.createParser(responseStream);

        while(!parser.isClosed()){
            JsonToken jsonToken = parser.nextToken();

            if(JsonToken.FIELD_NAME.equals(jsonToken)){
                String fieldName = parser.getCurrentName();
                jsonToken = parser.nextToken();

                if("access_token".equals(fieldName)){
                    String accesstoken = parser.getValueAsString();
                    System.out.println("Access Token: " +
accesstoken.substring(0,5)+ "..." +
accesstoken.substring(accesstoken.length()-5));
                    return;
                }
            }
        }
    }
}
```

Get a token using Go

Go

```
package main

import (
  "fmt"
  "io/ioutil"
  "net/http"
  "net/url"
  "encoding/json"
```

```
)

type responseJson struct {
  AccessToken string `json:"access_token"`
  RefreshToken string `json:"refresh_token"`
  ExpiresIn string `json:"expires_in"`
  ExpiresOn string `json:"expires_on"`
  NotBefore string `json:"not_before"`
  Resource string `json:"resource"`
  TokenType string `json:"token_type"`
}

func main() {

    // Create HTTP request for a managed services for Azure resources token
to access Azure Resource Manager
    var msi_endpoint *url.URL
    msi_endpoint, err :=
url.Parse("http://169.254.169.254/metadata/identity/oauth2/token?api-
version=2018-02-01")
    if err != nil {
      fmt.Println("Error creating URL: ", err)
      return
    }
    msi_parameters := msi_endpoint.Query()
    msi_parameters.Add("resource", "https://management.azure.com/")
    msi_endpoint.RawQuery = msi_parameters.Encode()
    req, err := http.NewRequest("GET", msi_endpoint.String(), nil)
    if err != nil {
      fmt.Println("Error creating HTTP request: ", err)
      return
    }
    req.Header.Add("Metadata", "true")

    // Call managed services for Azure resources token endpoint
    client := &http.Client{}
    resp, err := client.Do(req)
    if err != nil{
      fmt.Println("Error calling token endpoint: ", err)
      return
    }

    // Pull out response body
    responseBytes,err := ioutil.ReadAll(resp.Body)
    defer resp.Body.Close()
    if err != nil {
      fmt.Println("Error reading response body : ", err)
      return
    }

    // Unmarshall response body into struct
    var r responseJson
    err = json.Unmarshal(responseBytes, &r)
    if err != nil {
      fmt.Println("Error unmarshalling the response:", err)
```

```
        return
    }

    // Print HTTP response and marshalled response body elements to console
    fmt.Println("Response status:", resp.Status)
    fmt.Println("access_token: ", r.AccessToken)
    fmt.Println("refresh_token: ", r.RefreshToken)
    fmt.Println("expires_in: ", r.ExpiresIn)
    fmt.Println("expires_on: ", r.ExpiresOn)
    fmt.Println("not_before: ", r.NotBefore)
    fmt.Println("resource: ", r.Resource)
    fmt.Println("token_type: ", r.TokenType)
}
```

Get a token using PowerShell

The following example demonstrates how to use the managed identities for Azure resources REST endpoint from a PowerShell client to:

1. Acquire an access token.
2. Use the access token to call an Azure Resource Manager REST API and get information about the VM. Be sure to substitute your subscription ID, resource group name, and virtual machine name for `<SUBSCRIPTION-ID>`, `<RESOURCE-GROUP>`, and `<VM-NAME>`, respectively.

Azure PowerShell

```
Invoke-WebRequest -Uri
'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-
01&resource=https%3A%2F%2Fmanagement.azure.com%2F' -Headers
@{Metadata="true"}
```

Example of how to parse the access token from the response:

Azure PowerShell

```
# Get an access token for managed identities for Azure resources
$response = Invoke-WebRequest -Uri
'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-
01&resource=https%3A%2F%2Fmanagement.azure.com%2F' `
                              -Headers @{Metadata="true"}
$content =$response.Content | ConvertFrom-Json
$access_token = $content.access_token
echo "The managed identities for Azure resources access token is
$access_token"

# Use the access token to get resource information for the VM
$vmInfoRest = (Invoke-WebRequest -Uri
```

```
'https://management.azure.com/subscriptions/<SUBSCRIPTION-
ID>/resourceGroups/<RESOURCE-
GROUP>/providers/Microsoft.Compute/virtualMachines/<VM-NAME>?api-
version=2017-12-01' -Method GET -ContentType "application/json" -Headers @{
Authorization ="Bearer $access_token"}).content
echo "JSON returned from call to get VM info:"
echo $vmInfoRest
```

Get a token using CURL

Bash

```
curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-
version=2018-02-01&resource=https%3A%2F%2Fmanagement.azure.com%2F' -H
Metadata:true -s
```

Example of how to parse the access token from the response:

Bash

```
response=$(curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-
version=2018-02-01&resource=https%3A%2F%2Fmanagement.azure.com%2F' -H
Metadata:true -s)
access_token=$(echo $response | python -c 'import sys, json; print
(json.load(sys.stdin)["access_token"])')
echo The managed identities for Azure resources access token is
$access_token
```

Token caching

The managed identities subsystem caches tokens but we still recommend that you implement token caching in your code. You should prepare for scenarios where the resource indicates that the token is expired.

On-the-wire calls to Microsoft Entra ID result only when:

- Cache miss occurs due to no token in the managed identities for Azure resources subsystem cache.
- The cached token is expired.

Error handling

The managed identities endpoint signals errors via the status code field of the HTTP response message header, as either 4xx or 5xx errors:

Expand table

Status Code	Error Reason	How To Handle
404 Not found.	IMDS endpoint is updating.	Retry with Exponential Backoff. See guidance below.
410	IMDS is going through updates	IMDS will be available within 70 seconds
429 Too many requests.	IMDS Throttle limit reached.	Retry with Exponential Backoff. See guidance below.
4xx Error in request.	One or more of the request parameters was incorrect.	Don't retry. Examine the error details for more information. 4xx errors are design-time errors.
5xx Transient error from service.	The managed identities for Azure resources subsystem or Microsoft Entra ID returned a transient error.	It's safe to retry after waiting for at least 1 second. If you retry too quickly or too often, IMDS and/or Microsoft Entra ID may return a rate limit error (429).
time out	IMDS endpoint is updating.	Retry with Exponential Backoff. See guidance later.

If an error occurs, the corresponding HTTP response body contains JSON with the error details:

Expand table

Element	Description
error	Error identifier.
error_description	Verbose description of error. **Error descriptions can change at any time. Do not write code that branches based on values in the error description.**

HTTP response reference

This section documents the possible error responses. A "200 OK" status is a successful response, and the access token is contained in the response body JSON, in the access_token element.

Expand table

Status code	Error	Error Description	Solution
400 Bad Request	invalid_resource	AADSTS50001: The application named *<URI>* wasn't found in the tenant named *<TENANT-ID>*. This message shows if the tenant administrator hasn't installed the application or no tenant user consented to it. You might have sent your authentication request to the wrong tenant.\	(Linux only)
400 Bad Request	bad_request_102	Required metadata header not specified	Either the `Metadata` request header field is missing from your request, or is formatted incorrectly. The value must be specified as `true`, in all lower case. See the "Sample request" in the preceding REST section for an example.
401 Unauthorized	unknown_source	Unknown Source *<URI>*	Verify that your HTTP GET request URI is formatted correctly. The `scheme:host/resource-path` portion must be specified as `http://localhost:50342/oauth2/token`. See the "Sample request" in the preceding REST section for an example.
	invalid_request	The request is missing a required parameter, includes an	

Status code	Error	Error Description	Solution
		invalid parameter value, includes a parameter more than once, or is otherwise malformed.	
	unauthorized_client	The client isn't authorized to request an access token using this method.	Caused by a request on a VM that doesn't have managed identities for Azure resources configured correctly. See Configure managed identities for Azure resources on a VM using the Azure portal if you need assistance with VM configuration.
	access_denied	The resource owner or authorization server denied the request.	
	unsupported_response_type	The authorization server doesn't support obtaining an access token using this method.	
	invalid_scope	The requested scope is invalid, unknown, or malformed.	
500 Internal server error	unknown	Failed to retrieve token from the Active directory. For details see logs in *<file path>*	Verify that the VM has managed identities for Azure resources enabled. See Configure managed identities for Azure resources on a VM using the Azure portal if you need assistance with VM configuration. Also verify that your HTTP GET request URI is formatted correctly, particularly the resource URI specified

Status code	Error	Error Description	Solution
			in the query string. See the "Sample request" in the preceding REST section for an example, or Azure services that support Microsoft Entra authentication for a list of services and their respective resource IDs.

> **Important**
>
> - IMDS isn't intended to be used behind a proxy and doing so is unsupported. For examples of how to bypass proxies, refer to the Azure Instance Metadata Samples.

Retry guidance

Retry if you receive a 404, 429, or 5xx error code (see Error handling). If you receive a 410 error, it indicates that IMDS is going through updates and will be available in a maximum of 70 seconds.

Throttling limits apply to the number of calls made to the IMDS endpoint. When the throttling threshold is exceeded, IMDS endpoint limits any further requests while the throttle is in effect. During this period, the IMDS endpoint returns the HTTP status code 429 ("Too many requests"), and the requests fail.

For retry, we recommend the following strategy:

Expand table

Retry strategy	Settings	Values	How it works
ExponentialBackoff	Retry count Min back-off Max back-off Delta back-off First fast retry	5 0 sec 60 sec 2 sec false	Attempt 1 - delay 0 sec Attempt 2 - delay ~2 sec Attempt 3 - delay ~6 sec Attempt 4 - delay ~14 sec Attempt 5 - delay ~30 sec

Resource IDs for Azure services

See Azure Services with managed identities support for a list of resources that support managed identities for Azure resources.

Next steps

- To enable managed identities for Azure resources on an Azure VM, see Configure managed identities for Azure resources on a VM using the Azure portal.

Feedback

How to use managed identities for Azure resources on an Azure VM for sign-in

Article • 02/28/2025

This article provides PowerShell and CLI script examples for sign-in using managed identities for Azure resources service principal, and guidance on important topics such as error handling.

> **Note**
>
> We recommend that you use the Azure Az PowerShell module to interact with Azure. See Install Azure PowerShell to get started. To learn how to migrate to the Az PowerShell module, see Migrate Azure PowerShell from AzureRM to Az.

Prerequisites

- If you're not familiar with the managed identities for Azure resources feature, see this overview. If you don't have an Azure account, sign up for a free account before you continue.

If you plan to use the Azure PowerShell or Azure CLI examples in this article, be sure to install the latest version of Azure PowerShell or Azure CLI.

> **Important**
>
> - All sample script in this article assumes the command-line client is running on a VM with managed identities for Azure resources enabled. Use the VM "Connect" feature in the Azure portal, to remotely connect to your VM. For details on enabling managed identities for Azure resources on a VM, see Configure managed identities for Azure resources on a VM using the Azure portal, or one of the variant articles (using PowerShell, CLI, a template, or an Azure SDK).
> - To prevent errors during resource access, the VM's managed identity must be given at least "Reader" access at the appropriate scope (the VM or higher) to allow Azure Resource Manager operations on the VM. See Assign managed

identities for Azure resources access to a resource using the Azure portal for details.

Overview

Managed identities for Azure resources provide a service principal object , which is created upon enabling managed identities for Azure resources on the VM. The service principal can be given access to Azure resources, and used as an identity by script/command-line clients for sign-in and resource access. Traditionally, in order to access secured resources under its own identity, a script client would need to:

- be registered and consented with Microsoft Entra ID as a confidential/web client application
- sign in under its service principal, using the app's credentials (which are likely embedded in the script)

With managed identities for Azure resources, your script client no longer needs to do either, as it can sign in under the managed identities for Azure resources service principal.

Azure CLI

The following script demonstrates how to:

1. Sign in to Microsoft Entra ID under the VM's managed identity for Azure resources service principal

2. Call Azure Resource Manager and get the VM's service principal ID. CLI takes care of managing token acquisition/use for you automatically. Be sure to substitute your virtual machine name for `<VM-NAME>`.

   Azure CLI

   ```
   az login --identity

   $spID=$(az resource list -n <VM-NAME> --query [*].identity.principalId
   --out tsv)
   echo The managed identity for Azure resources service principal ID is
   $spID
   ```

Azure PowerShell

The following script demonstrates how to:

1. Sign in to Microsoft Entra ID under the VM's managed identity for Azure resources service principal
2. Call an Azure Resource Manager cmdlet to get information about the VM. PowerShell takes care of managing token use for you automatically.

Azure PowerShell

```
Add-AzAccount -identity

# Call Azure Resource Manager to get the service principal ID for the
VM's managed identity for Azure resources.
$vmInfoPs = Get-AzVM -ResourceGroupName <RESOURCE-GROUP> -Name <VM-
NAME>
$spID = $vmInfoPs.Identity.PrincipalId
echo "The managed identity for Azure resources service principal ID is
$spID"
```

Resource IDs for Azure services

See Azure services that support Microsoft Entra authentication for a list of resources that support Microsoft Entra ID and have been tested with managed identities for Azure resources, and their respective resource IDs.

Error handling guidance

Responses such as the following may indicate that the VM's managed identity for Azure resources has not been correctly configured:

- PowerShell: *Invoke-WebRequest : Unable to connect to the remote server*
- CLI: *MSI: Failed to retrieve a token from* `http://localhost:50342/oauth2/token` *with an error of 'HTTPConnectionPool(host='localhost', port=50342)*

If you receive one of these errors, return to the Azure VM in the Azure portal and go to the **Identity** page and ensure **System assigned** is set to "Yes."

Next steps

- To enable managed identities for Azure resources on an Azure VM, see Configure managed identities for Azure resources on an Azure VM using PowerShell, or Configure managed identities for Azure resources on an Azure VM using Azure CLI

Feedback

Was this page helpful? Yes No

Provide product feedback

How to use managed identities for Azure resources on an Azure VM with Azure SDKs

Article • 02/28/2025

This article provides a list of SDK samples, which demonstrate use of their respective Azure SDK's support for managed identities for Azure resources.

Prerequisites

- If you're not familiar with the managed identities for Azure resources feature, see this overview. If you don't have an Azure account, sign up for a free account before you continue.

> **Important**
>
> - All sample code/script in this article assumes the client is running on a VM with managed identities for Azure resources enabled. Use the VM "Connect" feature in the Azure portal, to remotely connect to your VM. For details on enabling managed identities for Azure resources on a VM, see **Configure managed identities for Azure resources on a VM using the Azure portal**, or one of the variant articles (using PowerShell, CLI, a template, or an Azure SDK).

SDK code samples

Expand table

SDK	Code sample
.NET	Deploy an Azure Resource Manager template from a Windows VM using managed identities for Azure resources
.NET Core	Call Azure services from a Linux VM using managed identities for Azure resources
Go	Azure identity client module for Go
Node.js	Manage resources using managed identities for Azure resources

SDK	Code sample
Python	Use managed identities for Azure resources to authenticate simply from inside a VM
Ruby	Manage resources from a VM with managed identities for Azure resources enabled

Next steps

- See Azure SDKs for the full list of Azure SDK resources, including library downloads, documentation, and more.
- To enable managed identities for Azure resources on an Azure VM, see Configure managed identities for Azure resources on a VM using the Azure portal.

Feedback

Was this page helpful? Yes No

Provide product feedback

Assign a managed identity access to an Azure resource or another resource

Article • 02/28/2025

This article shows you how to give an Azure virtual machine (VM) managed identity access to an Azure storage account. Once you've configured an Azure resource with a managed identity, you can then give the managed identity access to another resource, similar to any security principal.

Prerequisites

- Be sure you've enabled managed identity on an Azure resource, such as an Azure virtual machine or Azure virtual machine scale set. For more information about managed identity for Azure resources, see What are managed identities for Azure resources?.
- Review the difference between a system-assigned and user-assigned managed identity.
- If you don't already have an Azure account, sign up for a free account before continuing.

Use Azure RBAC to assign a managed identity access to another resource using the Azure portal

> **Important**
>
> The steps outlined below show is how you grant access to a service using Azure RBAC. Check specific service documentation on how to grant access; for example, check Azure Data Explorer for instructions. Some Azure services are in the process of adopting Azure RBAC on the data plane.

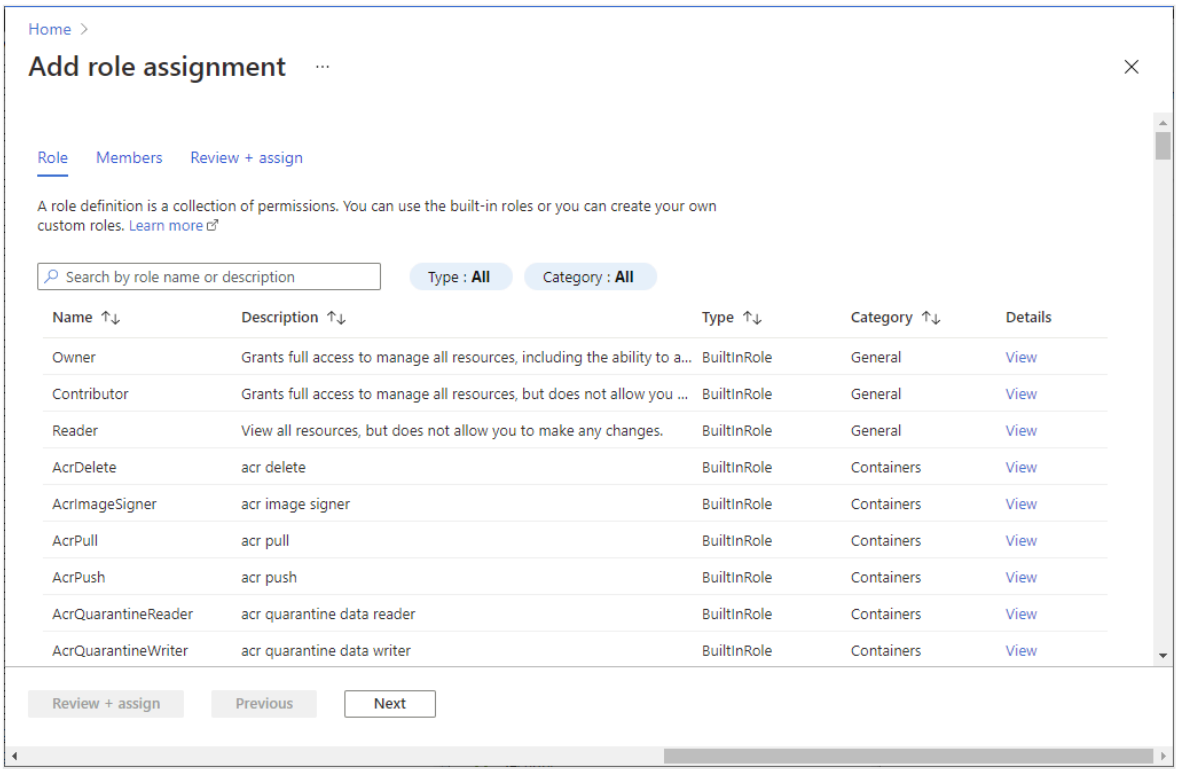
1. Sign in to the Azure portal using an account associated with the Azure subscription for which you've configured the managed identity.

2. Navigate to the desired resource that you want to modify access control. In this example, you'll give an Azure virtual machine (VM) access to a storage account, then you can navigate to the storage account.

3. Select **Access control (IAM)**.

4. Select **Add** > **Add role assignment** to open the **Add role assignment** page.

5. Select the role and managed identity. For detailed steps, see Assign Azure roles using the Azure portal.



Next steps

- To enable managed identities on an Azure virtual machine, see Configure managed identities for Azure resources.
- To enable managed identities on an Azure virtual machine scale set, see Configure managed identities for Azure resources on a virtual machine scale set.

Feedback

Was this page helpful? Yes No

Provide product feedback

Assign a managed identity access to an application role

Article • 03/14/2025

Managed identities for Azure resources provide Azure services with an identity in Microsoft Entra ID. They work without needing credentials in your code. Azure services use this identity to authenticate to services that support Microsoft Entra authentication. Application roles provide a form of role-based access control, and allow a service to implement authorization rules.

> **Note**
>
> The tokens your application receives are cached by the underlying infrastructure. This means that any changes to the managed identity's roles can take significant time to process. For more information, see Limitation of using managed identities for authorization.

In this article, you'll learn how to assign a managed identity to an application role exposed by another application using the Microsoft Graph PowerShell SDK or Azure CLI.

Prerequisites

- If you're unfamiliar with managed identities for Azure resources, see Managed identity for Azure resources overview.
- Review the difference between a system-assigned and user-assigned managed identity.
- If you don't already have an Azure account, sign up for a free account before continuing.

Assign a managed identity access to another application's app role using PowerShell

To run the example scripts, you have two options:

- Use the Azure Cloud Shell, which you can open using the **Try It** button on the top-right corner of code blocks.
- Run scripts locally by installing the latest version of the Microsoft Graph PowerShell SDK.

1. Enable managed identity on an Azure resource, such as an Azure VM.

2. Find the object ID of the managed identity's service principal.

   For a system-assigned managed identity, you can find the object ID on the Azure portal on the resource's **Identity** page. You can also use the following PowerShell script to find the object ID. You'll need the resource ID of the resource you created in step 1, which is available in the Azure portal on the resource's **Properties** page.

   PowerShell

   ```
   $resourceIdWithManagedIdentity = '/subscriptions/{my subscription ID}/resourceGroups/{my resource group name}/providers/Microsoft.Compute/virtualMachines/{my virtual machine name}'
   (Get-AzResource -ResourceId $resourceIdWithManagedIdentity).Identity.PrincipalId
   ```

   For a user-assigned managed identity, you can find the managed identity's object ID on the Azure portal on the resource's **Overview** page. You can also use the following PowerShell script to find the object ID. You'll need the resource ID of the user-assigned managed identity.

   PowerShell

   ```
   $userManagedIdentityResourceId = '/subscriptions/{my subscription ID}/resourceGroups/{my resource group name}/providers/Microsoft.ManagedIdentity/userAssignedIdentities/{my managed identity name}'
   (Get-AzResource -ResourceId $userManagedIdentityResourceId).Properties.PrincipalId
   ```

3. Create a new application registration to represent the service that you want your managed identity to send a request to.

   - If the API or service that exposes the app role grant to the managed identity already has a service principal in your Microsoft Entra tenant, skip this step. For example, in the case that you want to grant the managed identity access to the Microsoft Graph API.

4. Find the object ID of the service application's service principal. You can find this using the Microsoft Entra admin center.

   - Sign in to the Microsoft Entra admin center. In the left nav blade, select **Identity**, then **Applications**, and then select **Enterprise applications**. Then find the application and look for the **Object ID**.

- You can also find the service principal's object ID by its display name using the following PowerShell script:

```PowerShell
$serverServicePrincipalObjectId = (Get-MgServicePrincipal -Filter "DisplayName eq '$applicationName'").Id
```

> **Note**
>
> Display names for applications are not unique, so you should verify that you obtain the correct application's service principal.

5. Add an app role to the application you created in the previous step. You can then create the role using the Azure portal or by using Microsoft Graph.

   - For example, you could add an app role by running the following query in Graph explorer:

```HTTP
PATCH /applications/{id}/

{
    "appRoles": [
        {
            "allowedMemberTypes": [
                "User",
                "Application"
            ],
            "description": "Read reports",
            "id": "00001111-aaaa-2222-bbbb-3333cccc4444",
            "displayName": "Report reader",
            "isEnabled": true,
            "value": "report.read"
        }
    ]
}
```

6. Assign the app role to the managed identity. You'll need the following information to assign the app role:

   - `managedIdentityObjectId`: the object ID of the managed identity's service principal, which you found in the previous step.
   - `serverServicePrincipalObjectId`: the object ID of the server application's service principal, which you found in step 4.

- `appRoleId`: the ID of the app role exposed by the server app, which you generated in step 5 - in the example, the app role ID is `00000000-0000-0000-0000-000000000000`.

- Execute the following PowerShell command to add the role assignment:

PowerShell

```
New-MgServicePrincipalAppRoleAssignment `
    -ServicePrincipalId $serverServicePrincipalObjectId `
    -PrincipalId $managedIdentityObjectId `
    -ResourceId $serverServicePrincipalObjectId `
    -AppRoleId $appRoleId
```

Complete example script

This example script shows you how to assign an Azure web app's managed identity to an app role.

PowerShell

```
# Install the module.
# Install-Module Microsoft.Graph -Scope CurrentUser

# Your tenant ID (in the Azure portal, under Azure Active Directory > Overview).
$tenantID = '<tenant-id>'

# The name of your web app, which has a managed identity that should be assigned to the server app's app role.
$webAppName = '<web-app-name>'
$resourceGroupName = '<resource-group-name-containing-web-app>'

# The name of the server app that exposes the app role.
$serverApplicationName = '<server-application-name>' # For example, MyApi

# The name of the app role that the managed identity should be assigned to.
$appRoleName = '<app-role-name>' # For example, MyApi.Read.All

# Look up the web app's managed identity's object ID.
$managedIdentityObjectId = (Get-AzWebApp -ResourceGroupName $resourceGroupName -Name $webAppName).identity.principalid

Connect-MgGraph -TenantId $tenantId -Scopes 'Application.Read.All','Application.ReadWrite.All','AppRoleAssignment.ReadWrite.All','Directory.AccessAsUser.All','Directory.Read.All','Directory.ReadWrite.All'

# Look up the details about the server app's service principal and app role.
```

```
$serverServicePrincipal = (Get-MgServicePrincipal -Filter "DisplayName eq
'$serverApplicationName'")
$serverServicePrincipalObjectId = $serverServicePrincipal.Id
$appRoleId = ($serverServicePrincipal.AppRoles | Where-Object {$_.Value -eq
$appRoleName }).Id

# Assign the managed identity access to the app role.
New-MgServicePrincipalAppRoleAssignment `
    -ServicePrincipalId $serverServicePrincipalObjectId `
    -PrincipalId $managedIdentityObjectId `
    -ResourceId $serverServicePrincipalObjectId `
    -AppRoleId $appRoleId
```

Next steps

- Managed identity for Azure resources overview
- To enable managed identity on an Azure VM, see Configure managed identities for Azure resources on an Azure VM.

Feedback

Was this page helpful? Yes No

Provide product feedback

Manage user-assigned managed identities

Article • 01/15/2025

Managed identities for Azure resources eliminate the need to manage credentials in code. You can use them to get a Microsoft Entra token for your applications. The applications can use the token when accessing resources that support Microsoft Entra authentication. Azure manages the identity so you don't have to.

There are two types of managed identities: system-assigned and user-assigned. System-assigned managed identities have their lifecycle tied to the resource that created them. This identity is restricted to only one resource, and you can grant permissions to the managed identity by using Azure role-based access control (RBAC). User-assigned managed identities can be used on multiple resources. To learn more about managed identities, see What are managed identities for Azure resources?.

In this article, you learn how to create, list, delete, or assign a role to a user-assigned managed identity by using the Azure portal.

Prerequisites

- If you're unfamiliar with managed identities for Azure resources, check out the overview section. Be sure to review the difference between a system-assigned and user-assigned managed identity.
- If you don't already have an Azure account, sign up for a free account before you continue.

Create a user-assigned managed identity

> **Tip**
>
> Steps in this article might vary slightly based on the portal you start from.

To create a user-assigned managed identity, your account needs the Managed Identity Contributor role assignment.

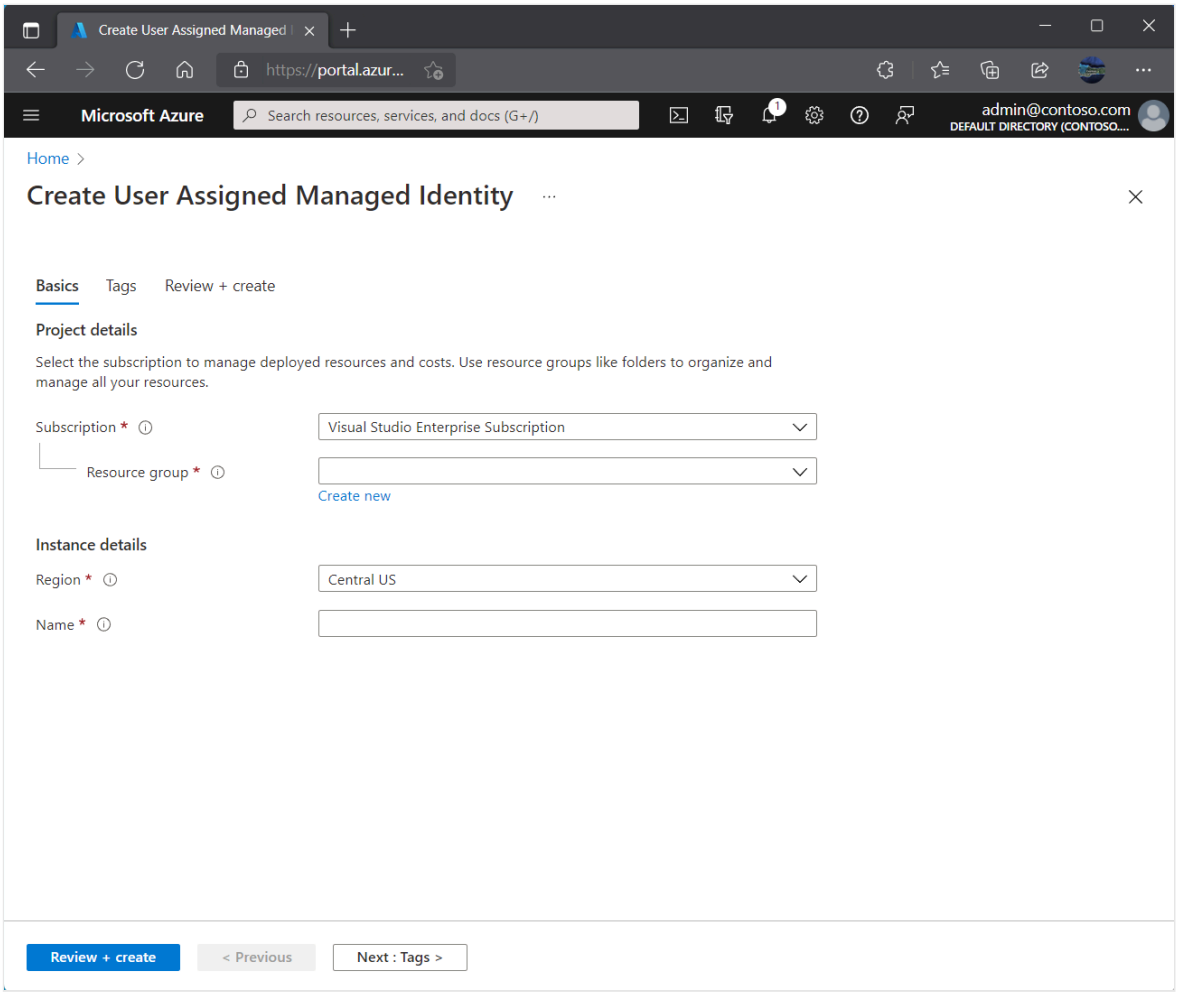
1. Sign in to the Azure portal.

2. In the search box, enter **Managed Identities**. Under **Services**, select **Managed Identities**.

3. Select **Add**, and enter values in the following boxes in the **Create User Assigned Managed Identity** pane:

    - **Subscription**: Choose the subscription to create the user-assigned managed identity under.
    - **Resource group**: Choose a resource group to create the user-assigned managed identity in, or select **Create new** to create a new resource group.
    - **Region**: Choose a region to deploy the user-assigned managed identity, for example, **West US**.
    - **Name**: Enter the name for your user-assigned managed identity, for example, UAI1.

    > **ⓘ Important**
    >
    > When you create user-assigned managed identities, the name must start with a letter or number, and may include a combination of alphanumeric characters, hyphens (-) and underscores (_). For the assignment to a virtual machine or virtual machine scale set to work properly, the name is limited to 24 characters. For more information, see **FAQs and known issues**.



4. Select **Review + create** to review the changes.

5. Select **Create**.

List user-assigned managed identities

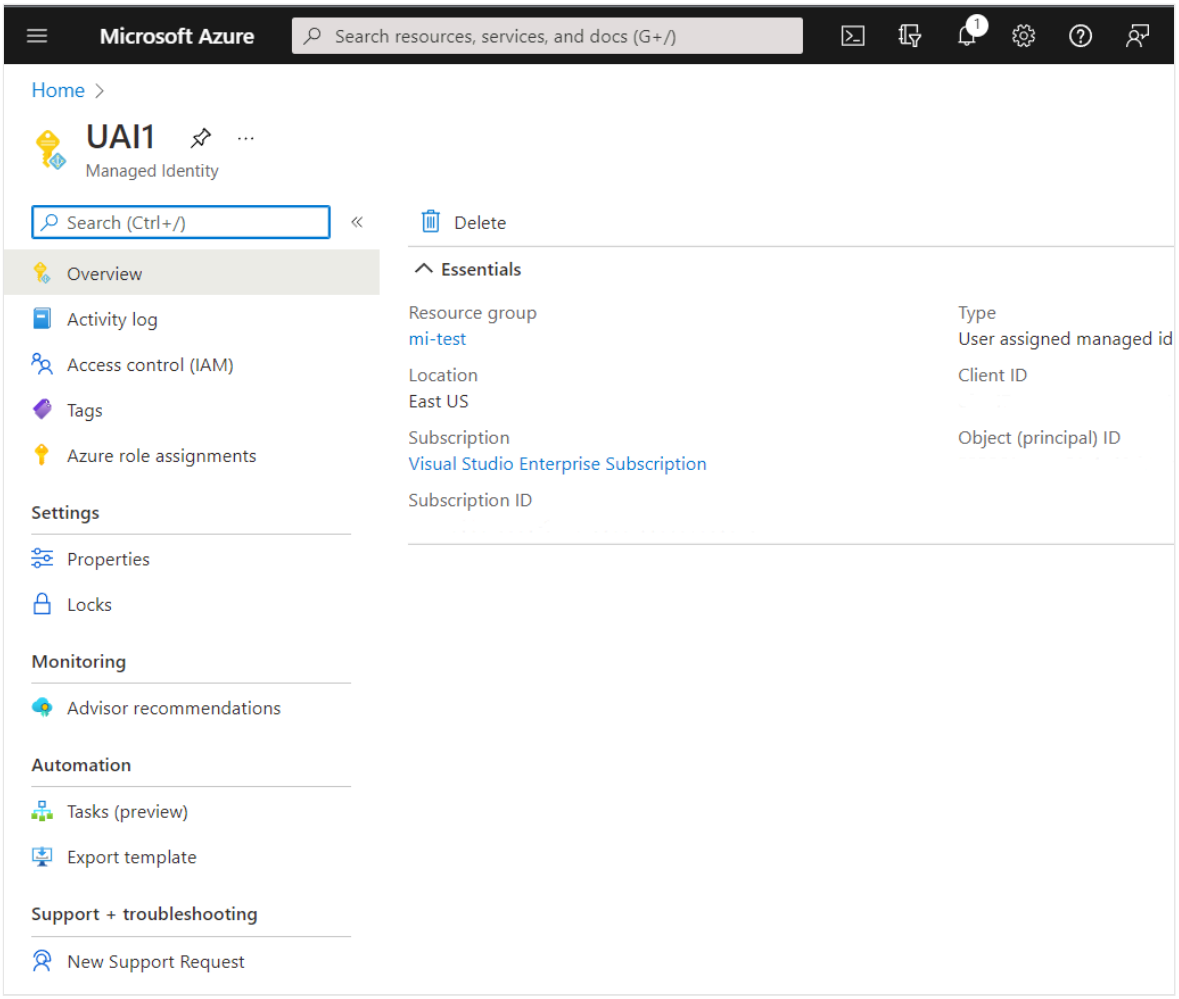
To list or read a user-assigned managed identity, your account needs to have either Managed Identity Operator or Managed Identity Contributor role assignments.

1. Sign in to the Azure portal.

2. In the search box, enter **Managed Identities**. Under **Services**, select **Managed Identities**.

3. A list of the user-assigned managed identities for your subscription is returned. To see the details of a user-assigned managed identity, select its name.

4. You can now view the details about the managed identity as shown in the image.

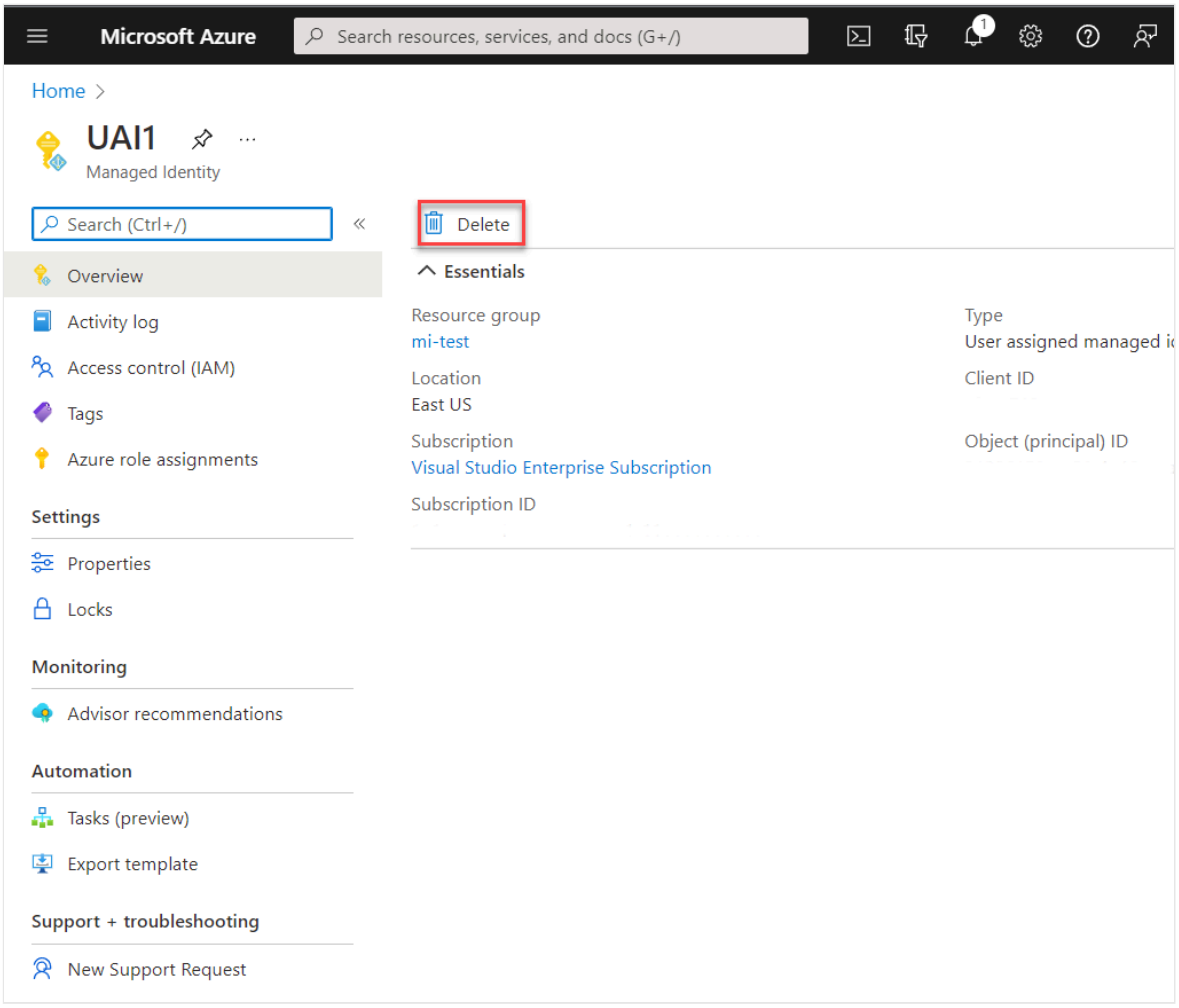


Delete a user-assigned managed identity

To delete a user-assigned managed identity, your account needs the Managed Identity Contributor role assignment.

Deleting a user-assigned identity doesn't remove it from the VM or resource it was assigned to. To remove the user-assigned identity from a VM, see Remove a user-assigned managed identity from a VM.

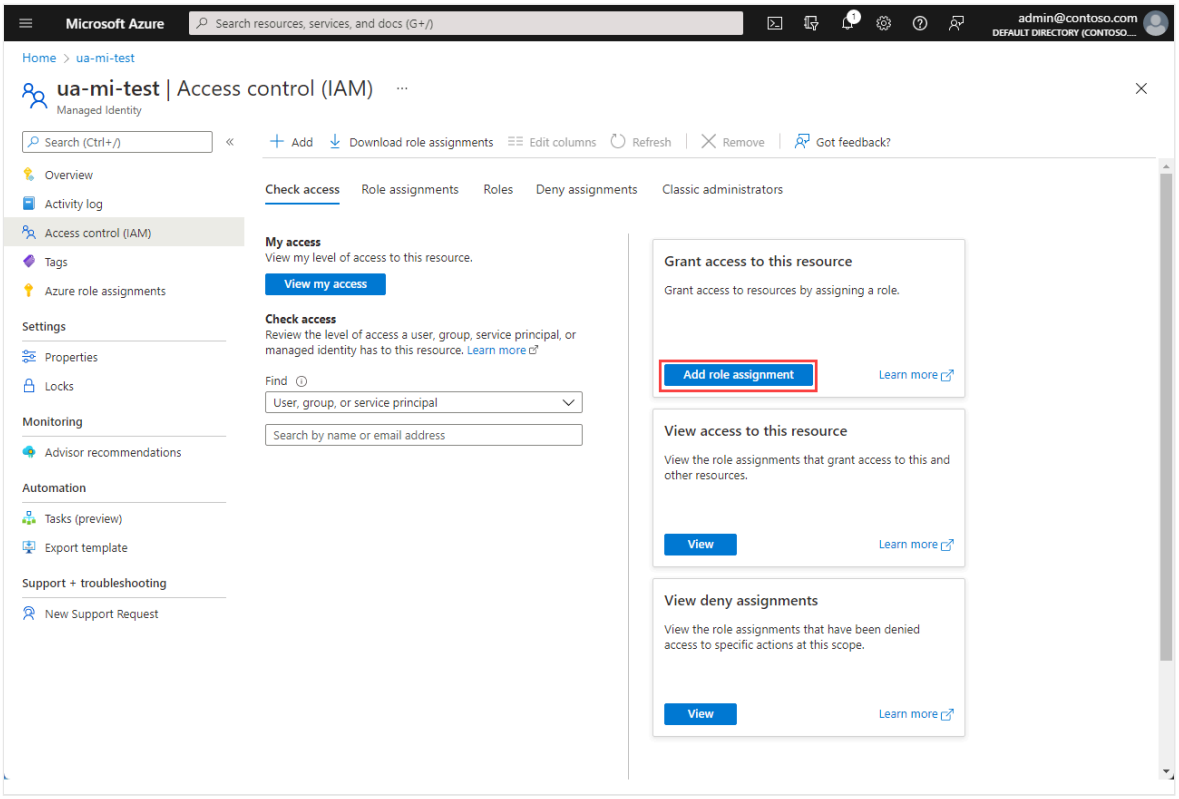
1. Sign in to the Azure portal .
2. Select the user-assigned managed identity, and select **Delete**.
3. Under the confirmation box, select **Yes**.



Manage access to user-assigned managed identities

In some environments, administrators choose to limit who can manage user-assigned managed identities. Administrators can implement this limitation using built-in RBAC roles. You can use these roles to grant a user or group in your organization rights over a user-assigned managed identity.

1. Sign in to the Azure portal.
2. In the search box, enter **Managed Identities**. Under **Services**, select **Managed Identities**.
3. A list of the user-assigned managed identities for your subscription is returned. Select the user-assigned managed identity that you want to manage.
4. Select **Access control (IAM)**.
5. Choose **Add role assignment**.



6. In the **Add role assignment** pane, choose the role to assign and choose **Next**.

7. Choose who should have the role assigned.

> **Note**
>
> You can find information on assigning roles to managed identities in Assign a managed identity access to a resource by using the Azure portal

Feedback

Was this page helpful? Yes No

Provide product feedback

View associated Azure resources for a user-assigned managed identity (Preview)

Article • 03/14/2025

This article explains how to view the Azure resources that are associated with a user-assigned managed identity. This feature is available in public preview.

Prerequisites

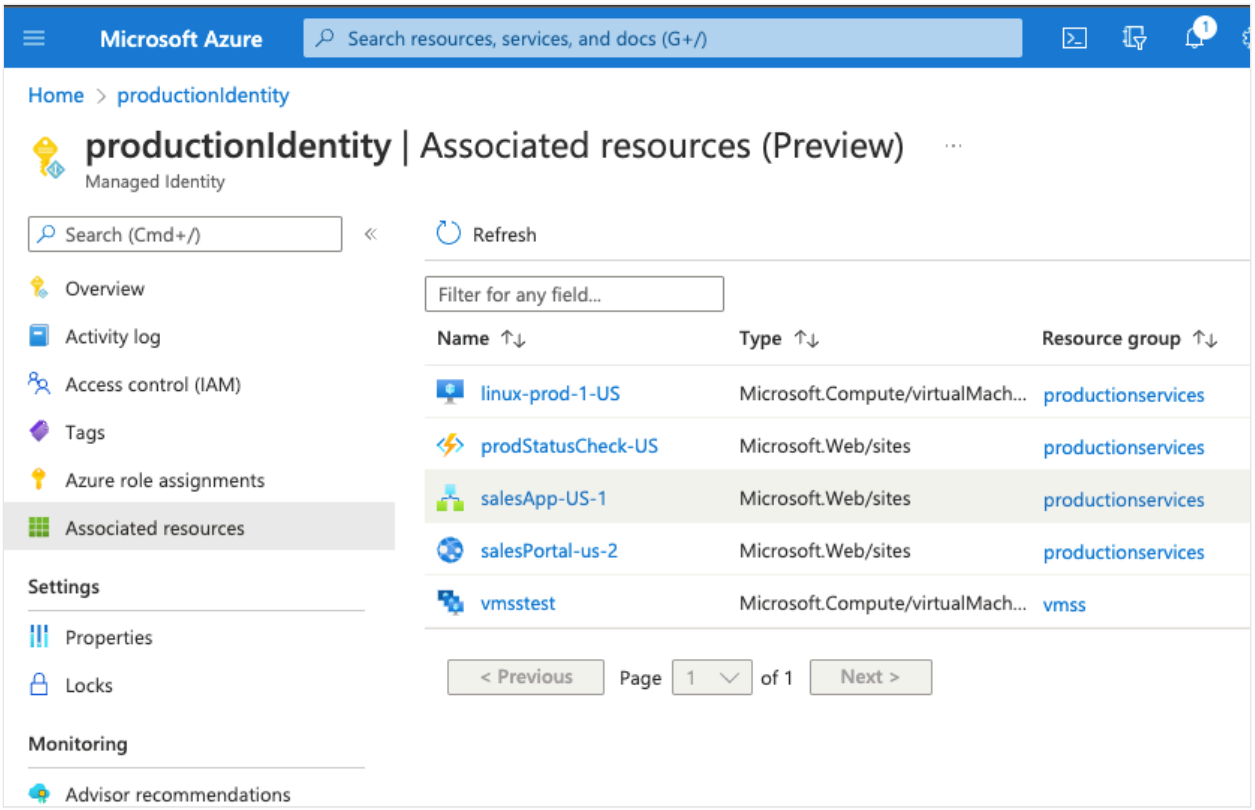
- If you're unfamiliar with managed identities for Azure resources, check out the overview section.
- If you don't already have an Azure account, sign up for a free account.

View resources for a user-assigned managed identity

Being able to quickly see which Azure resources are associated with a user-assigned managed identity gives you greater visibility into your environment. You can quickly identify unused identities that can be safely deleted, and know which resources are affected by changing the permissions or group membership of a managed identity.

Portal

- From the **Azure portal** search for **Managed Identities**.
- Select a managed identity
- In the left-hand menu, select the **Associated resources** link
- A list of the Azure resources associated with the managed identity is displayed



Select the resource name to be brought to its summary page.

Filtering and sorting by resource type

Filter the resources by typing in the filter box at the top of the summary page. You can filter by the name, type, resource group, and subscription ID.

Select the column title to sort alphabetically, ascending or descending.

REST API

The list of associated resources can also be accessed using the REST API. This endpoint is separate to the API endpoint used to retrieve a list of user-assigned managed identities. You need the following information:

- Subscription ID
- Resource name of the user-assigned managed identity that you want to view the resources for
- Resource group of the user-assigned managed identity

Request format

```
https://management.azure.com/subscriptions/{resourceID of user-assigned
identity}/listAssociatedResources?$filter={filter}&$orderby={orderby}&$skip=
```

```
{skip}&$top={top}&$skiptoken={skiptoken}&api-version=2021-09-30-preview
```

Parameters

Expand table

Parameter	Example	Description
$filter	`type eq 'microsoft.cognitiveservices/account' and contains(name, 'test')`	An OData expression that allows you to filter any of the available fields: name, type, resourceGroup, subscriptionId, subscriptionDisplayName The following operations are supported: `and`, `or`, `eq` and `contains`
$orderby	`name asc`	An OData expression that allows you to order by any of the available fields
$skip	50	The number of items you want to skip while paging through the results.
$top	10	The number of resources to return. Zero returns only a count of the resources.

You can see a sample request to the REST API:

HTTP

```
POST https://management.azure.com/subscriptions/aaaa0a0a-bb1b-cc2c-dd3d-eeeeee4e4e4e/resourceGroups/devrg/providers/Microsoft.ManagedIdentity/userAssignedIdentities/devIdentity/listAssociatedResources?$filter={filter}&$orderby={orderby}&$skip={skip}&$top={top}&skipToken={skipToken}&api-version=2021-09-30-preview
```

Notice a sample response from the REST API:

JSON

```
{
  "totalCount": 2,
  "value": [
    {
      "id": "/subscriptions/{subId}/resourceGroups/testrg/providers/Microsoft.CognitiveServices/accounts/test1",
      "name": "test1",
```

```
      "type": "microsoft.cognitiveservices/accounts",
      "resourceGroup": "testrg",
      "subscriptionId": "{subId}",
      "subscriptionDisplayName": "TestSubscription"
    },
    {
      "id":
"/subscriptions/{subId}/resourceGroups/testrg/providers/Microsoft.CognitiveS
ervices/accounts/test2",
      "name": "test2",
      "type": "microsoft.cognitiveservices/accounts",
      "resourceGroup": "testrg",
      "subscriptionId": "{subId}",
      "subscriptionDisplayName": "TestSubscription"
    }
  ],
  "nextLink":
"https://management.azure.com/subscriptions/{subId}/resourceGroups/testrg/pr
oviders/Microsoft.ManagedIdentity/userAssignedIdentities/testid?
skiptoken=ew0KICAiJGlkIjogIjEiLA0KICAiTWF4Um93cyI6IDIsDQogICJSb3dzVG9Ta2lwIj
ogMiwNCiAgIkt1c3RvQ2x1c3RlclVybCI6ICJodHRwczovL2FybXRvcG9sb2d5Lmt1c3RvLndpbm
Rvd3MubmV0Ig0KfQ%253d%253d&api-version=2021"
}
```


Command Line Interface

To view the associated resources for a user-assigned managed identity, run the following command:

Azure CLI

```
az identity list-resources --resource-group <ResourceGroupName> --name
<ManagedIdentityName>
```

The response looks like this:

JSON


```
[
  {
    "id": "/subscriptions/XXXX-XXXX-XXXX-XXXX-
XXXfc47ab8130/resourceGroups/ProductionServices/providers/Microsoft.Compute/
virtualMachines/linux-prod-1-US",
    "name": "linux-prod-1-US",
    "resourceGroup": "productionservices",
    "subscriptionDisplayName": "Visual Studio Enterprise Subscription",
    "subscriptionId": "XXXX-XXXX-XXXX-XXXX-XXXfc47ab8130",
    "type": "microsoft.compute/virtualmachines"
  },
```

```
  {
    "id": "/subscriptions/XXXX-XXXX-XXXX-XXXX-
XXXfc47ab8130/resourceGroups/ProductionServices/providers/Microsoft.Web/site
s/prodStatusCheck-US",
    "name": "prodStatusCheck-US",
    "resourceGroup": "productionservices",
    "subscriptionDisplayName": "Visual Studio Enterprise Subscription",
    "subscriptionId": "XXXX-XXXX-XXXX-XXXX-XXXfc47ab8130",
    "type": "microsoft.web/sites"
  },
  {
    "id": "/subscriptions/XXXX-XXXX-XXXX-XXXX-
XXXfc47ab8130/resourceGroups/ProductionServices/providers/Microsoft.Web/site
s/salesApp-US-1",
    "name": "salesApp-US-1",
    "resourceGroup": "productionservices",
    "subscriptionDisplayName": "Visual Studio Enterprise Subscription",
    "subscriptionId": "XXXX-XXXX-XXXX-XXXX-XXXfc47ab8130",
    "type": "microsoft.web/sites"
  },
  {
    "id": "/subscriptions/XXXX-XXXX-XXXX-XXXX-
XXXfc47ab8130/resourceGroups/ProductionServices/providers/Microsoft.Web/site
s/salesPortal-us-2",
    "name": "salesPortal-us-2",
    "resourceGroup": "productionservices",
    "subscriptionDisplayName": "Visual Studio Enterprise Subscription",
    "subscriptionId": "XXXX-XXXX-XXXX-XXXX-XXXfc47ab8130",
    "type": "microsoft.web/sites"
  },
  {
    "id": "/subscriptions/XXXX-XXXX-XXXX-XXXX-
XXXfc47ab8130/resourceGroups/vmss/providers/Microsoft.Compute/virtualMachine
ScaleSets/vmsstest",
    "name": "vmsstest",
    "resourceGroup": "vmss",
    "subscriptionDisplayName": "Visual Studio Enterprise Subscription",
    "subscriptionId": "XXXX-XXXX-XXXX-XXXX-XXXfc47ab8130",
    "type": "microsoft.compute/virtualmachinescalesets"
  }
]
```


REST API using PowerShell

There's no specific PowerShell command for returning the associated resources of a managed identity, but you can use the REST API in PowerShell by using the following command:

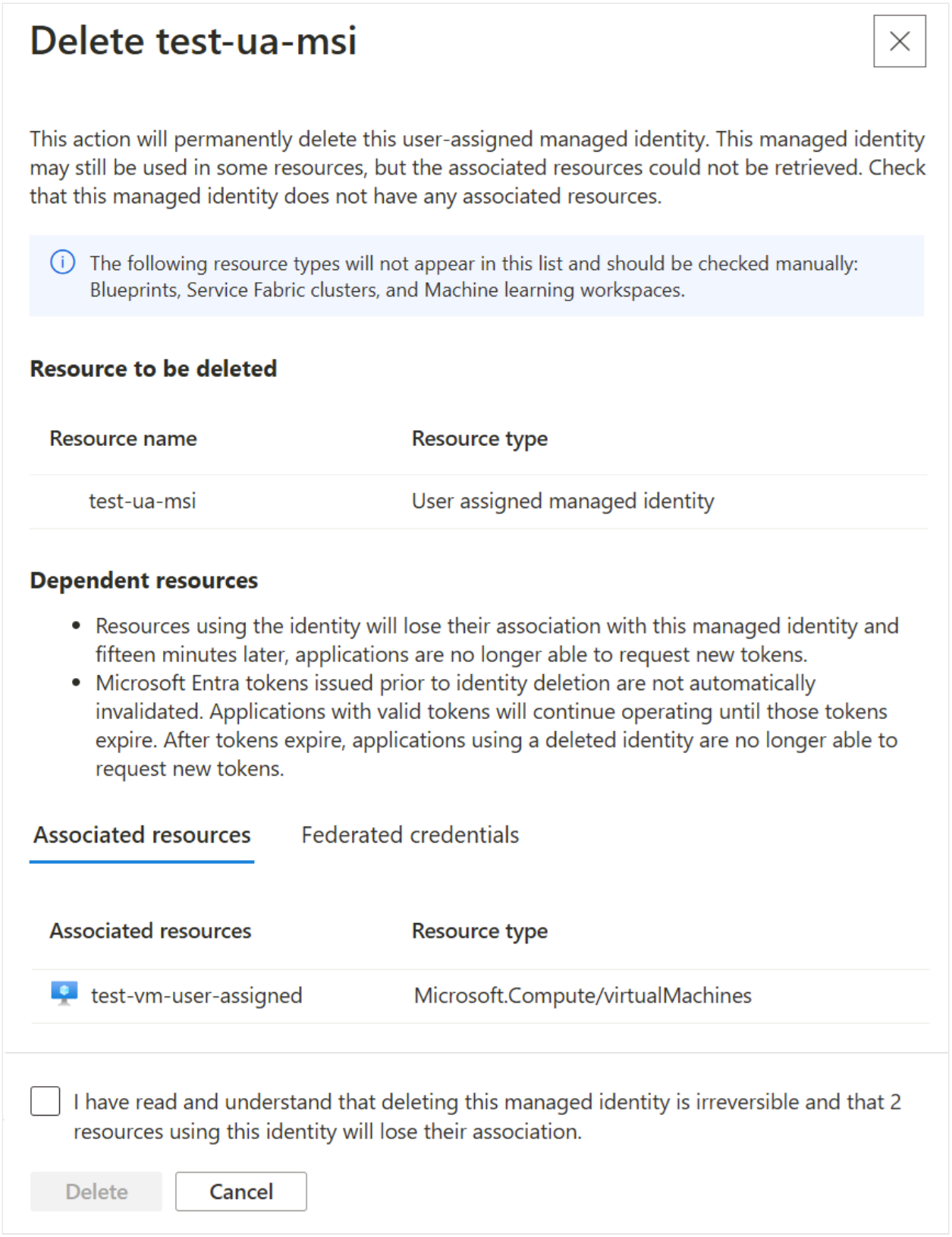
```
PowerShell
```

```
Invoke-AzRestMethod -Path "/subscriptions/XXX-XXX-XXX-
XXX/resourceGroups/test-
rg/providers/Microsoft.ManagedIdentity/userAssignedIdentities/test-identity-
name/listAssociatedResources?api-version=2021-09-30-
PREVIEW&%24orderby=name%20asc&%24skip=0&%24top=100" -Method Post
```

> **Note**
>
> All resources associated with an identity are returned, regardless of the user's permissions. The user only needs to have access to read the managed identity. This means that more resources may be visible than the user can see elsewhere in the portal. This is to provide full visibility of the identity's usage. If the user doesn't have access to an associated resource, an error is displayed if they try to access it from the list.

Delete a user-assigned managed identity

When you select the delete button for a user-assigned managed identity, you see a list of up to 10 associated resources for that identity. The full count is displayed at the top of the pane. This list allows you to see which resources are affected by deleting the identity. You're asked to confirm your decision.



This confirmation process is only available in the portal. To view an identity's resources before deleting it using the REST API, retrieve the list of resources manually in advance.

Limitations

- This functionality is available in all public regions and in USGov and China.
- API requests for associated resources are limited to one per second per tenant. If you exceed this limit, you may receive an `HTTP 429` error. This limit doesn't apply

to retrieving a list of user-assigned managed identities.

- Azure Resources types that are in preview, or their support for Managed identities is in preview, may not appear in the associated resources list until fully generally available. This list includes Service Fabric clusters, Blueprints, and Machine learning services.
- This functionality is limited to tenants with fewer than 5,000 subscriptions. An error is displayed if the tenant has greater than 5,000 subscriptions.
- The list of associated resources display the resource type, not the display name.
- Azure Policy assignments appear in the list, but their names aren't displayed correctly.
- This functionality isn't yet available through PowerShell.

Next steps

- Managed identities for Azure resources

Feedback

View the service principal of a managed identity

Article • 03/14/2025

Managed identities for Azure resources provide Azure services with an automatically managed identity in Microsoft Entra ID. You can use this identity to authenticate to any service that supports Microsoft Entra authentication without having credentials in your code.

In this article, you'll learn how to view the service principal of a managed identity.

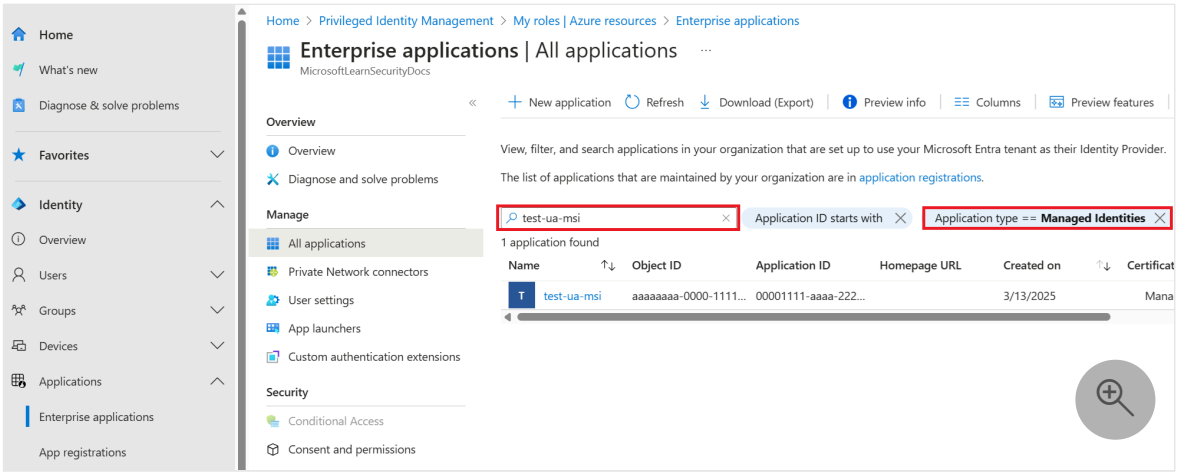
> **Note**
>
> Service principals are enterprise applications.

Prerequisites

- If you're unfamiliar with managed identities for Azure resources, see What are managed identities for Azure resources?.
- If you don't already have an Azure account, sign up for a free account before continuing.
- Enable system assigned identity on a virtual machine or application.

View the service principal for a managed identity using the Azure portal

1. Sign in to the Microsoft Entra admin center.
2. In the left nav blade, select **Identity**, then **Applications**, and then select **Enterprise applications**.
3. In the **Manage** section select **All applications**.
4. Set a filter for "Application type == Managed Identities" and select **Apply**.
5. (Optional) In the search filter box, enter the name of the Azure resource that has system managed identities enabled or the name of the user assigned managed identity.



Next steps

For more information about managed identities, see Managed identities for Azure resources.

Feedback

Was this page helpful? Yes No

Provide product feedback

View update and sign-in activities for Managed identities

Article • 06/06/2024

This article explains how to view updates carried out to managed identities, and sign-in attempts made by managed identities.

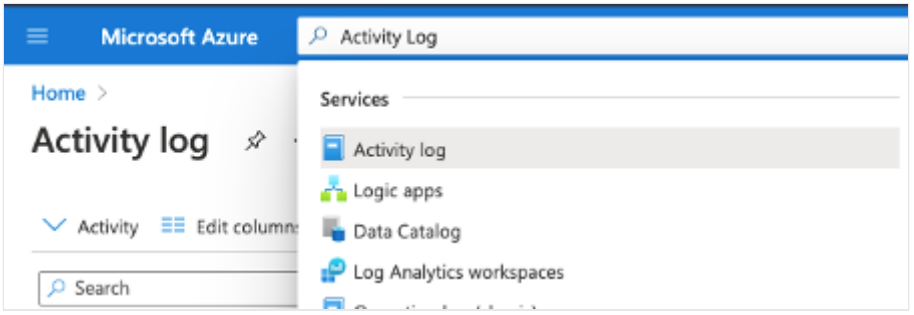
Prerequisites

- If you're unfamiliar with managed identities for Azure resources, check out the overview section.
- If you don't already have an Azure account, sign up for a free account.

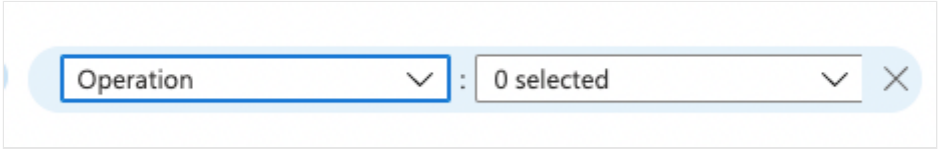
View updates made to user-assigned managed identities

This procedure demonstrates how to view updates carried out to user-assigned managed identities.

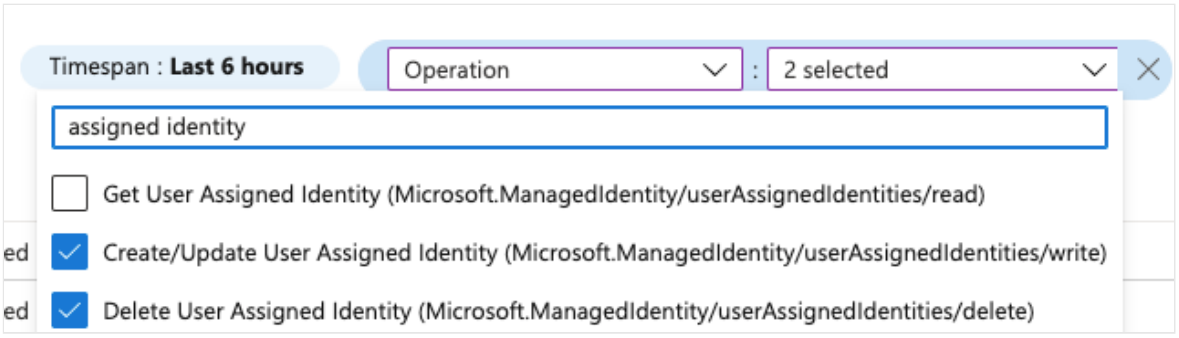
1. In the Azure portal, browse to **Activity Log**.



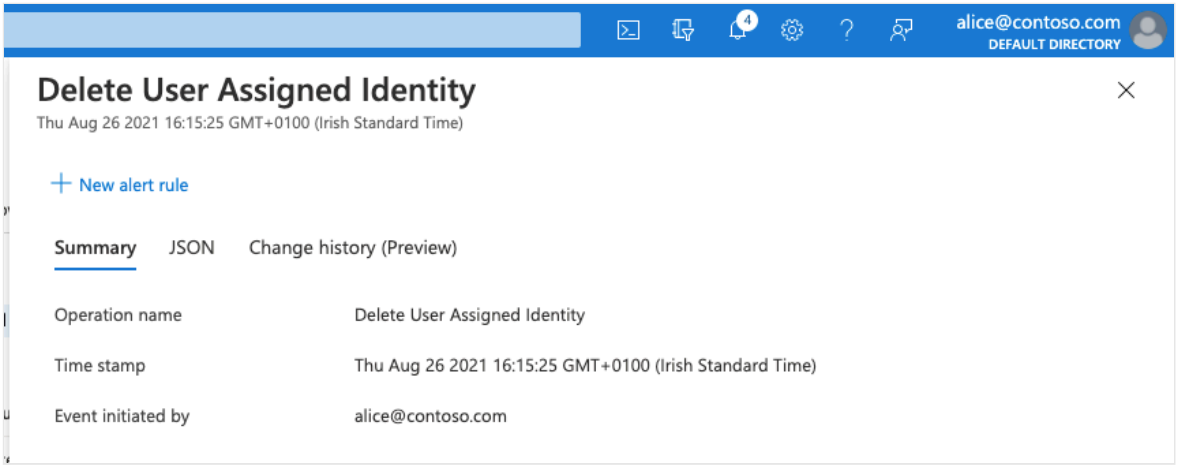
2. Select the **Add Filter** search pill and select **Operation** from the list.



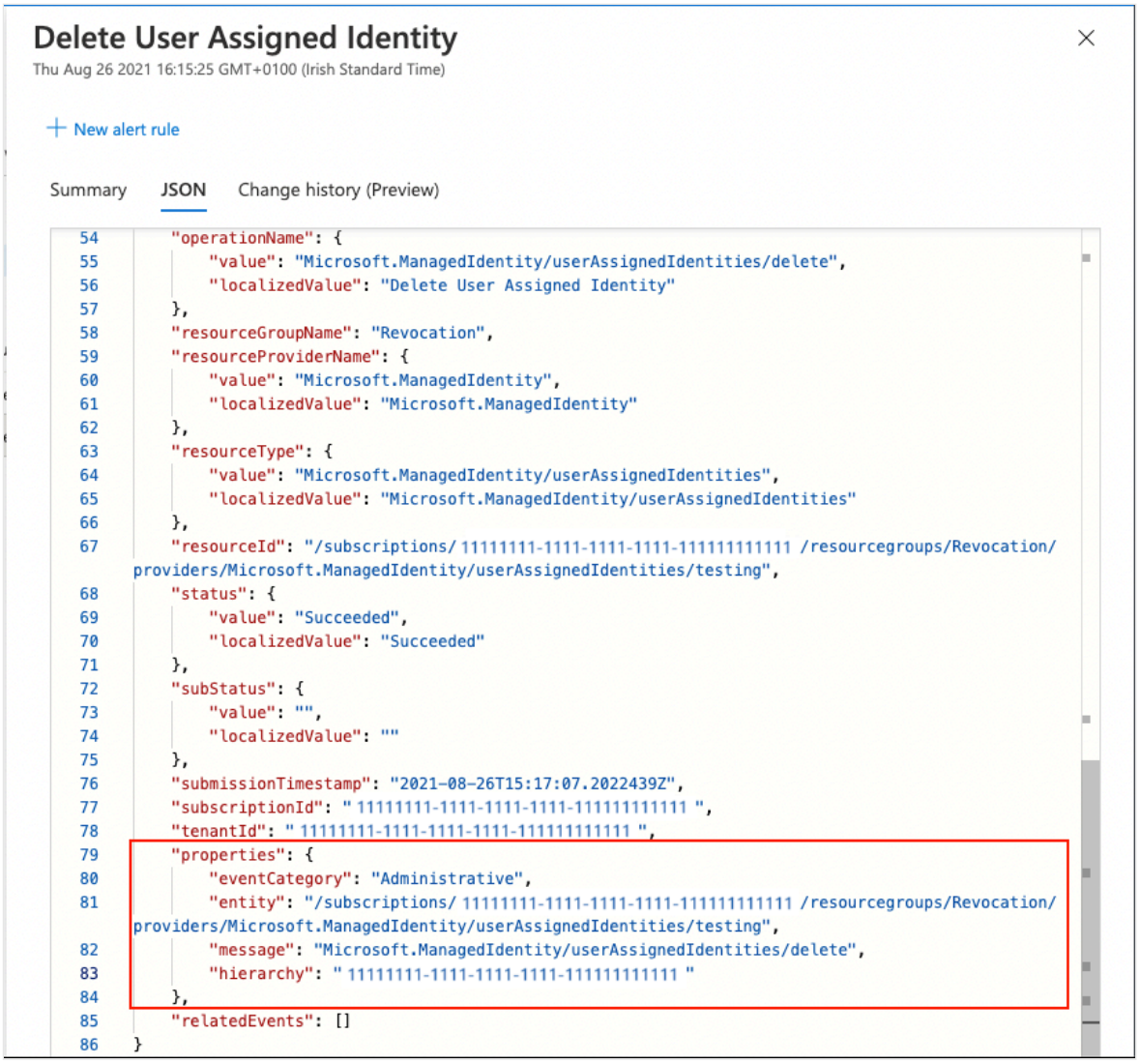
3. In the **Operation** dropdown list, enter these operation names: "Delete User Assigned Identity" and "Write UserAssignedIdentities".



4. When matching operations are displayed, select one to view the summary.



5. Select the **JSON** tab to view more detailed information about the operation, and scroll to the **properties** node to view information about the identity that was modified.

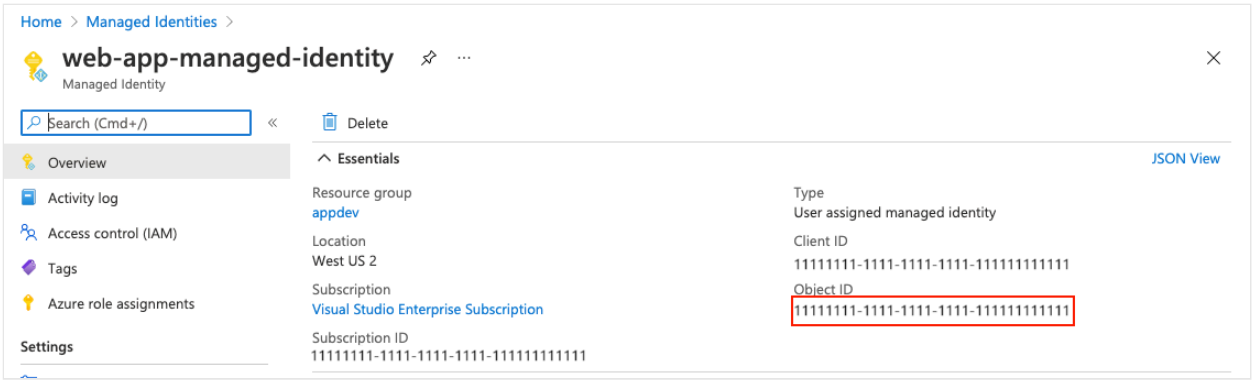


View role assignments added and removed for managed identities

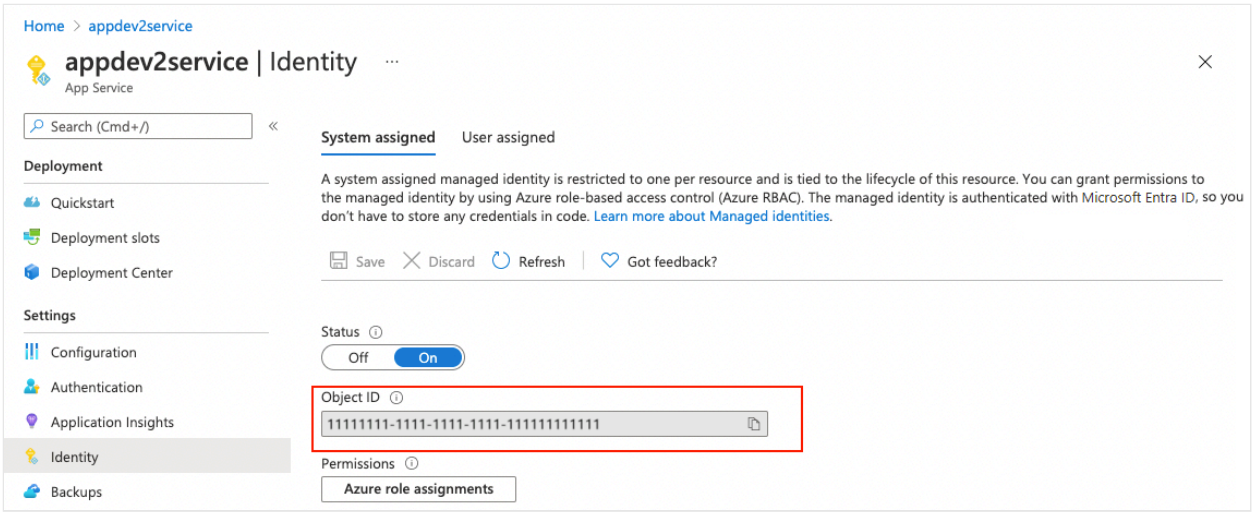
> **Note**
>
> You'll need to search by the object (principal) ID of the managed identity that you want to view role assignment changes for.

1. Locate the managed identity you wish to view the role assignment changes for. If you're looking for a system-assigned managed identity, the object ID is displayed in the **Identity** screen under the resource. If you're looking for a user-assigned identity, the object ID is displayed in the **Overview** page of the managed identity.

User-assigned identity:

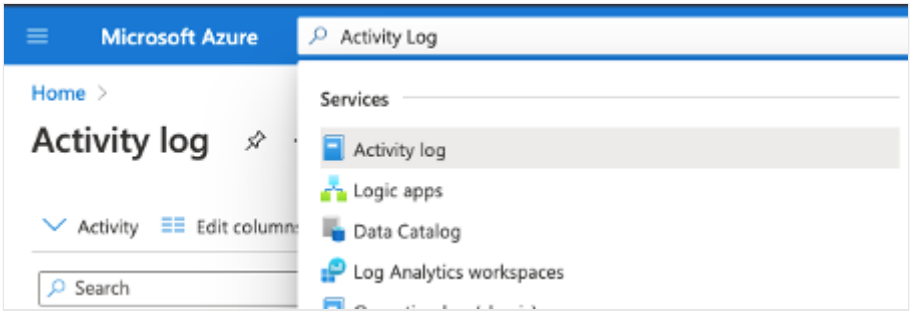


System-assigned identity:

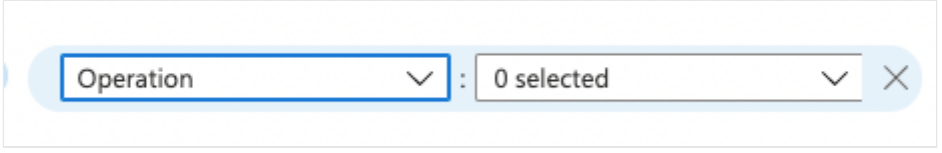


1. Copy the object ID.

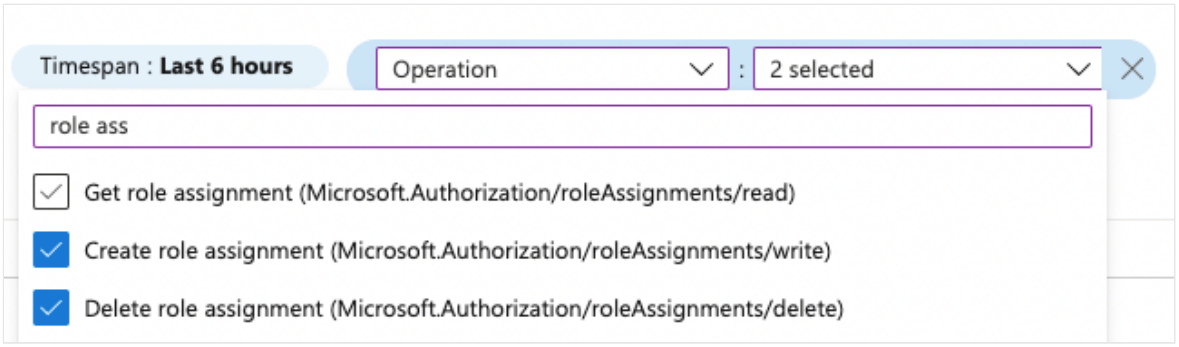
2. Browse to the **Activity log**.



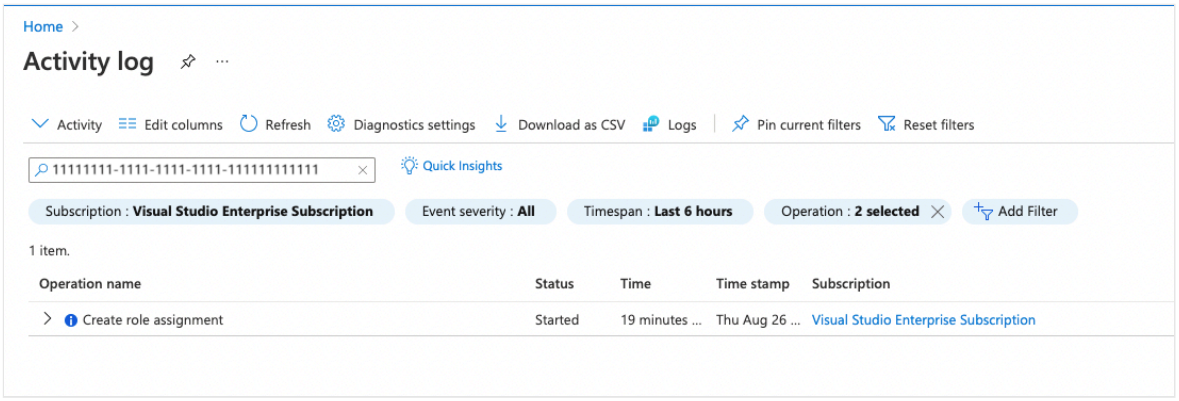
3. Select the **Add Filter** search pill and select **Operation** from the list.



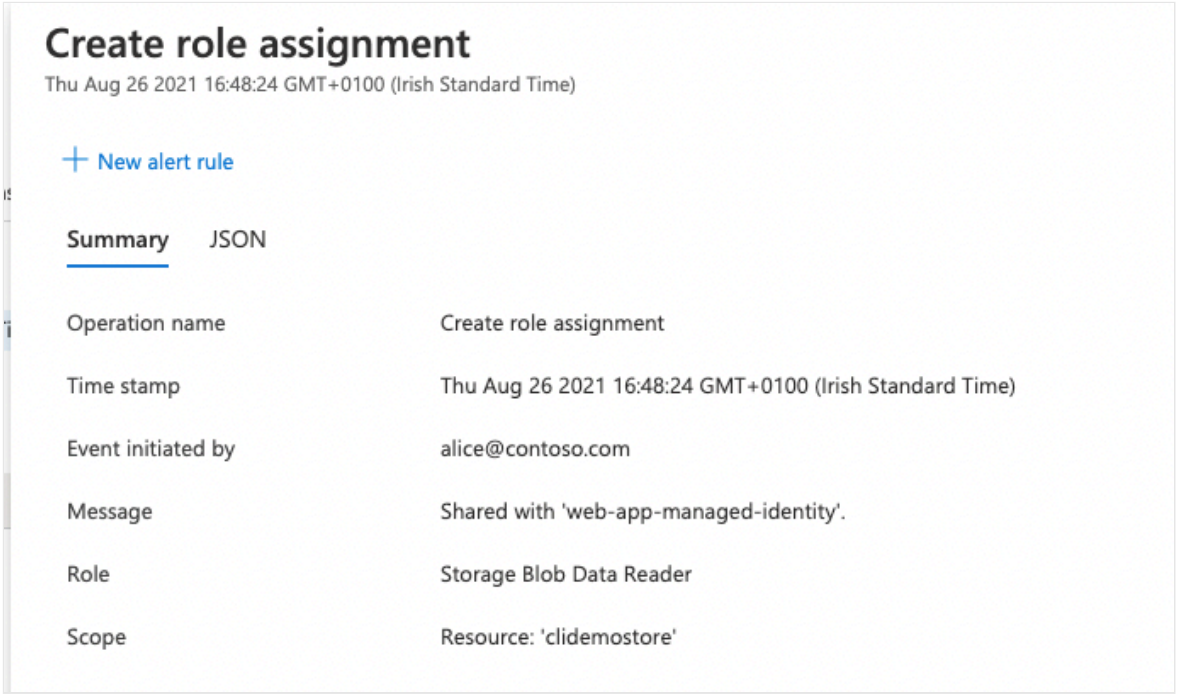
4. In the **Operation** dropdown list, enter these operation names: **Create role assignment** and **Delete role assignment**.



5. Paste the object ID in the search box; the results are filtered automatically.

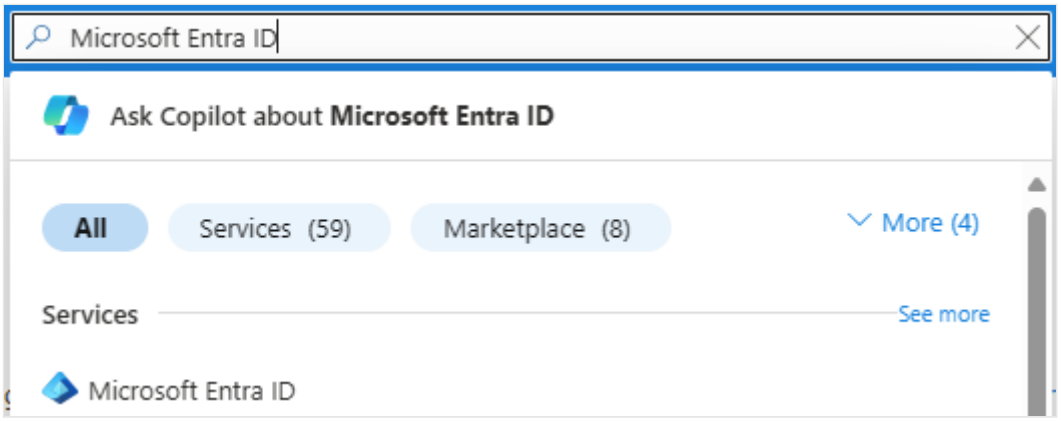


6. When matching operations are displayed, select one to view the summary.

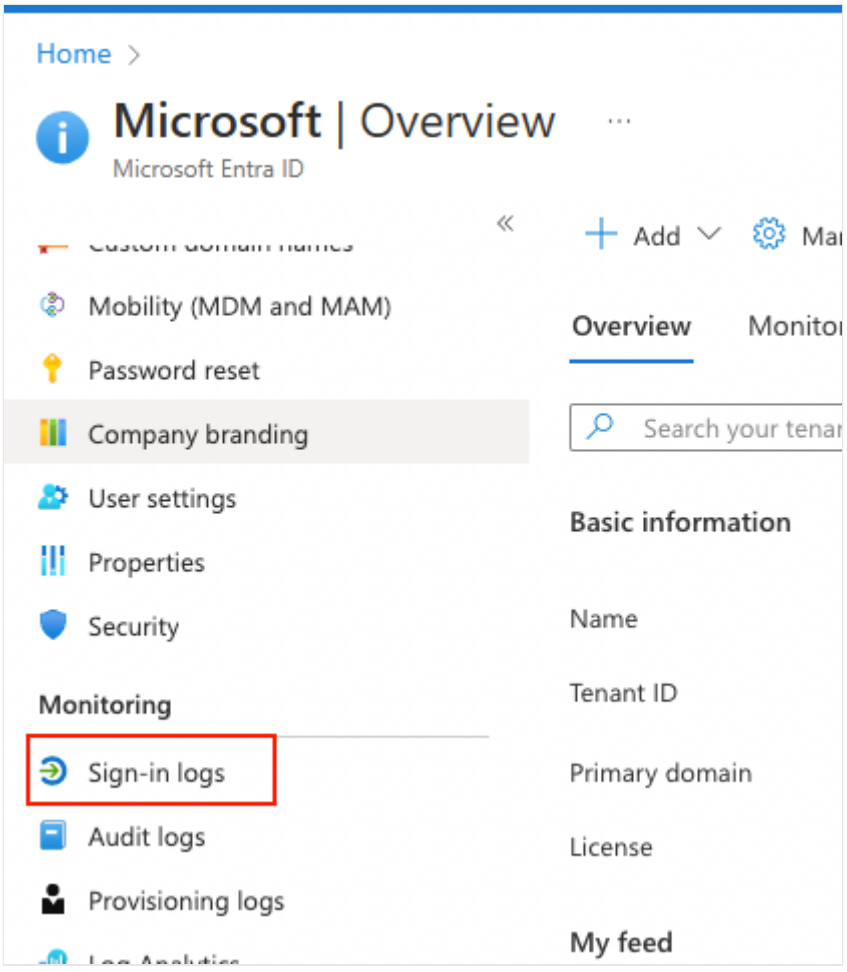


View authentication attempts by managed identities

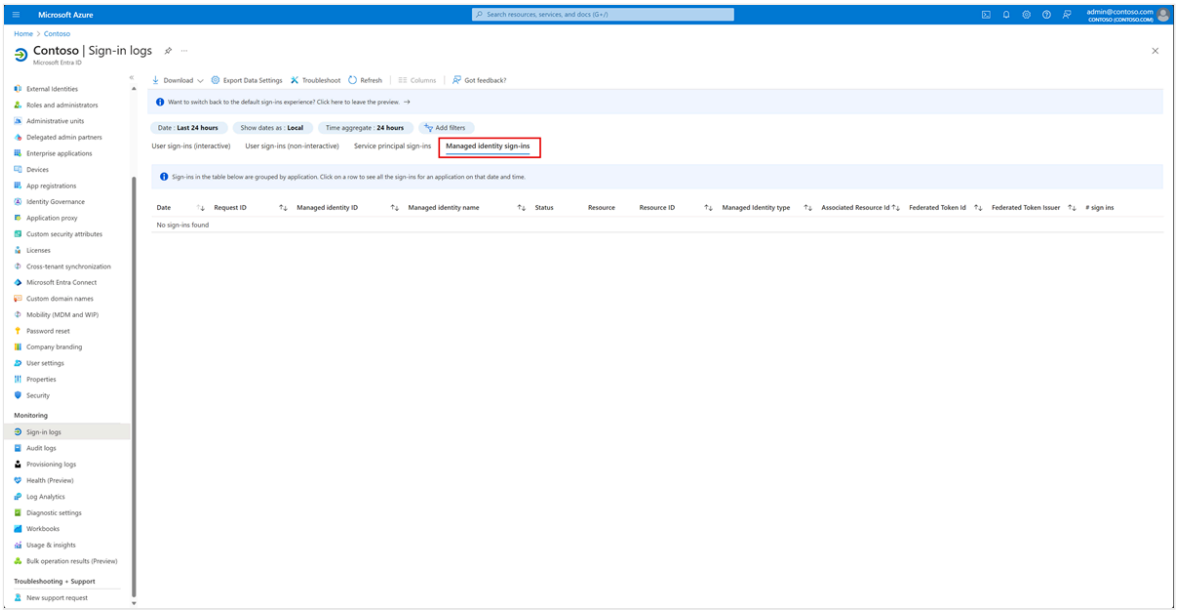
1. Browse to **Microsoft Entra ID**.



2. Select **Sign-in logs** from the **Monitoring** section.

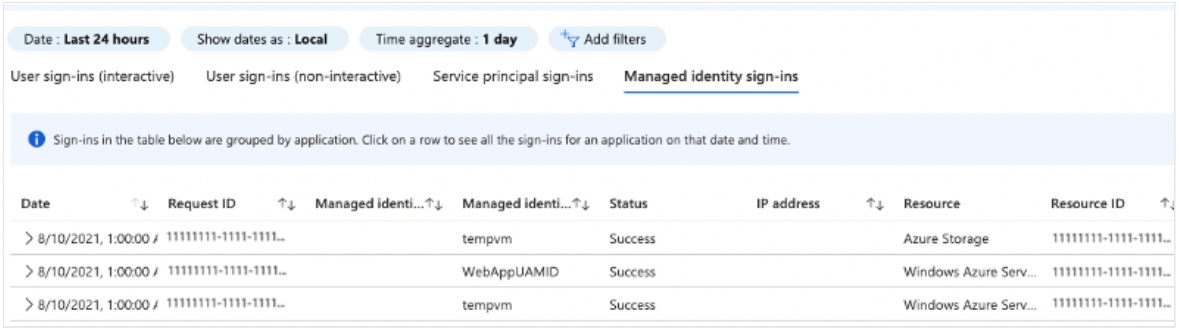


3. Select the **Managed identity sign-ins** tab.



4. To view the identity's Enterprise application in Microsoft Entra ID, select the "Managed Identity ID" column.

5. To view the Azure resource or user-assigned managed identity, search by name in the search bar of the Azure portal.



> **Note**
>
> Since managed identity authentication requests originate within the Azure infrastructure, the IP Address value is excluded here.

Next steps

- Managed identities for Azure resources
- Azure Activity log
- Microsoft Entra sign-in log

Feedback

Was this page helpful? Yes No

Provide product feedback

Move managed identity for Azure resources across regions

Article • 10/23/2023

There are situations in which you'd want to move your existing user-assigned managed identities from one region to another. For example, you may need to move a solution that uses user-assigned managed identities to another region. You may also want to move an existing identity to another region as part of disaster recovery planning, and testing.

Moving User-assigned managed identities across Azure regions isn't supported. You can however, recreate a user-assigned managed identity in the target region.

Prerequisites

- Permissions to list permissions granted to existing user-assigned managed identity.
- Permissions to grant a new user-assigned managed identity the required permissions.
- Permissions to assign a new user-assigned identity to the Azure resources.
- Permissions to edit Group membership, if your user-assigned managed identity is a member of one or more groups.

Prepare and move

1. Copy user-assigned managed identity assigned permissions. You can list Azure role assignments but that may not be enough depending on how permissions were granted to the user-assigned managed identity. You should confirm that your solution doesn't depend on permissions granted using a service specific option.
2. Create a new user-assigned managed identity at the target region.
3. Grant the managed identity the same permissions as the original identity that it's replacing, including Group membership. You can review Assign Azure roles to a managed identity, and Group membership.
4. Specify the new identity in the properties of the resource instance that uses the newly created user assigned managed identity.

Verify

After reconfiguring your service to use your new managed identities in the target region, you need to confirm that all operations have been restored.

Clean up

Once that you confirm your service is back online, you can proceed to delete any resources in the source region that you no longer use.

Next steps

In this tutorial, you took the steps needed to recreate a user-assigned managed identity in a new region.

- Manage user-assigned managed identities

Client libraries for managed identities authentication

Article • 12/10/2024

This document provides an overview of the client libraries available for authenticating your applications using managed identities for Azure resources. These libraries include the Azure Identity libraries and Microsoft Authentication Libraries (MSAL).

Some Azure services built client libraries on top of these libraries. For example, the `Microsoft.Data.SqlClient` package can be used to authenticate to an Azure SQL database using managed identities. Behind the scenes, the Azure Identity library for .NET is being used.

Choosing the right library

MSAL libraries offer lower-level abstractions than libraries like Azure Identity. Both MSAL and Azure Identity libraries allow you to acquire tokens via managed identity. Internally, Azure Identity libraries use MSAL and provide higher-level APIs such as `DefaultAzureCredential` that remove the need to implement manual switches between identity types when developing and deploying your application.

- If your application already uses one of the libraries, continue using the same library.
- If you're developing a new application and plan to call other Azure resources, use an Azure Identity library. This library provides an improved developer experience by allowing the app to authenticate on local developer machines where managed identities are not available.
- If you need to call other downstream web APIs like Microsoft Graph or your own web API, use MSAL. For .NET applications, use the *Microsoft.Identity.Web* library, which is built on top of MSAL.

In cases where an Azure service built a client library on top of these libraries, consider using the service-specific client library. For example, for Azure SQL, use the Microsoft.Data.SqlClient package.

Language-specific API references

Expand table

Language	Azure Identity	MSAL
.NET	Azure Identity client library for .NET	MSAL .NET
C++	Azure Identity client library for C++	
Java	Azure Identity client library for Java	MSAL Java
JavaScript	Azure Identity client library for JavaScript	MSAL JavaScript
Python	Azure Identity client library for Python	MSAL Python
Go	Azure Identity client library for Go	MSAL Go

Feedback

Was this page helpful? Yes No

Provide product feedback

Managed identities for Azure resources frequently asked questions

Article • 02/27/2025

Administration

How can you find resources that have a managed identity?

You can find the list of resources that have a system-assigned managed identity by using the following Azure CLI Command:

Azure CLI

```
az resource list --query "[?identity.type=='SystemAssigned'].{Name:name,
principalId:identity.principalId}" --output table
```

Which Azure role-based access control (RBAC) permissions are required to use a managed identity on a resource?

- System-assigned managed identity: You need to have write permissions over the resource. For example, for virtual machines you need `Microsoft.Compute/virtualMachines/write`. This action is included in resource specific built-in roles like Virtual Machine Contributor.
- Assigning user-assigned managed identities to resources: You need write permissions over the resource. For example, for virtual machines you need `Microsoft.Compute/virtualMachines/write`. You need `Microsoft.ManagedIdentity/userAssignedIdentities/*/assign/action` action over the user-assigned identity. This action is included in the Managed Identity Operator built-in role.
- Managing user-assigned identities: To create or delete user-assigned managed identities, you need the Managed Identity Contributor role assignment.
- Managing role assignments for managed identities: You need the Owner or User Access Administrator role assignment over the resource to which you're granting access. You'll need the Reader role assignment to the resource with a system-assigned identity, or to the user-assigned identity that is being given the role

assignment. If you don't have read access, you can search by "User, group, or service principal" to find the identity's backing service principal, instead of searching by managed identity while adding the role assignment. Read more about assigning Azure roles.

How do I prevent the creation of user-assigned managed identities?

You can keep your users from creating user-assigned managed identities using Azure Policy

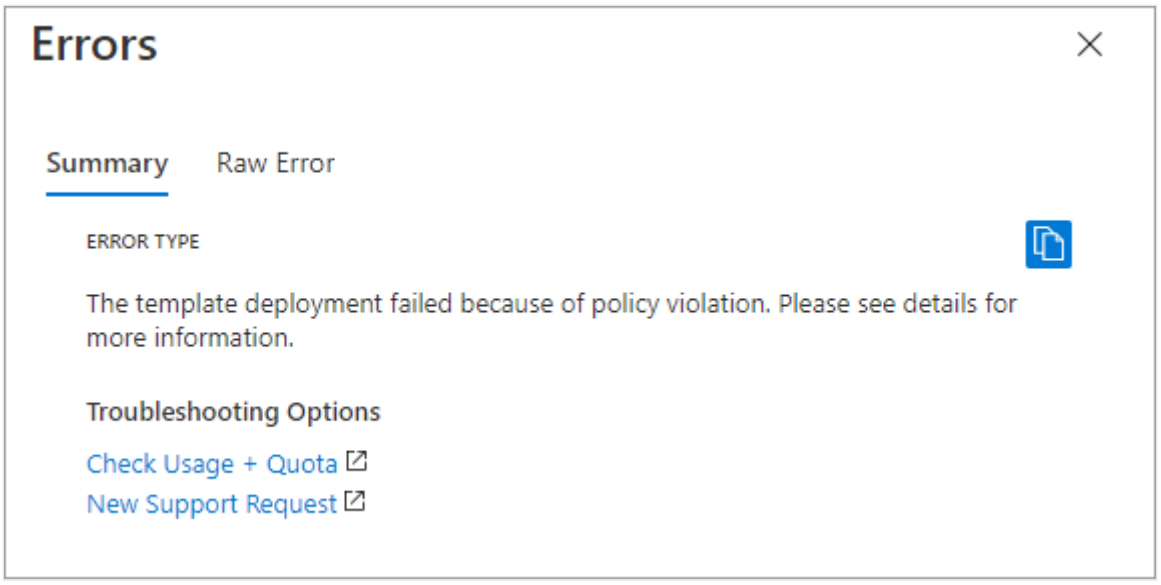
1. Sign in to the Azure portal and go to **Policy**.
2. Choose **Definitions**
3. Select **+ Policy definition** and enter the necessary information.
4. In the policy rule section, paste:

```JSON
{
  "mode": "All",
  "policyRule": {
    "if": {
      "field": "type",
      "equals": "Microsoft.ManagedIdentity/userAssignedIdentities"
    },
    "then": {
      "effect": "deny"
    }
  },
  "parameters": {}
}
```

After creating the policy, assign it to the resource group that you would like to use.

1. Navigate to resource groups.
2. Find the resource group that you're using for testing.
3. Choose **Policies** from the left menu.
4. Select **Assign policy**
5. In the **Basics** section, provide:
   a. **Scope** The resource group that we're using for testing
   b. **Policy definition**: The policy that we created earlier.
6. Leave all other settings at their defaults and choose **Review + Create**

At this point, any attempt to create a user-assigned managed identity in the resource group fails.



Concepts

Do managed identities have a backing app object?

No, managed identities and Microsoft Entra App Registrations aren't the same thing in the directory.

App registrations have two components: an application object and a service principal object. A managed identity only has a service principal object.

Managed identities don't have an application object in the directory, which is what is commonly used to grant app permissions for Microsoft Graph. Instead, Microsoft Graph permissions for managed identities need to be granted directly to the service principal.

What is the credential associated with a managed identity? How long is it valid and how often is it rotated?

> **Note**
>
> How managed identities authenticate is an internal implementation detail that is subject to change without notice.

Managed identities use certificate-based authentication. Each managed identity's credential has an expiration of 90 days and it's rolled after 45 days.

What identity will IMDS default to if I don't specify the identity in the request?

- If system assigned managed identity is enabled and no identity is specified in the request, Azure Instance Metadata Service (IMDS) defaults to the system assigned managed identity.
- If system assigned managed identity isn't enabled, and only one user assigned managed identity exists, IMDS defaults to that single user assigned managed identity.

  > If another user assigned managed identity is assigned to the resource for any reason, your requests to IMDS will start failing with the error `Multiple user assigned identities exist, please specify the clientId / resourceId of the identity in the token request`. We highly recommend you explicitly specify an identity in your request, even if only one user assigned managed identity currently exists for the resource.

- If system assigned managed identity isn't enabled, and multiple user assigned managed identities exist, then you're required to specify a managed identity in the request.

Limitations

Can the same managed identity be used across multiple regions?

In short, yes you can use user assigned managed identities in more than one Azure region. The longer answer is that while user assigned managed identities are created as regional resources the associated service principal (SP) created in Microsoft Entra ID is available globally. The service principal can be used from any Azure region and its availability is dependent on the availability of Microsoft Entra ID. For example, if you created a user assigned managed identity in the South-Central region and that region becomes unavailable this issue only impacts control plane activities on the managed identity itself. The activities performed by any resources already configured to use the managed identities wouldn't be impacted.

Does managed identities for Azure resources work with Azure Cloud Services (Classic)?

Managed identities for Azure resources don't have support for Azure Cloud Services (classic) at this time.

What is the security boundary of managed identities for Azure resources?

The security boundary of the identity is the resource to which it's attached. For example, the security boundary for a virtual machine with managed identities for Azure resources enabled, is the virtual machine. Any code running on that VM, is able to call the managed identities endpoint and request tokens. The experience is similar experience when working with other resources that support managed identities.

Will managed identities be recreated automatically if I move a subscription to another directory?

No, if you move a subscription to another directory, you have to manually re-create them and grant Azure role assignments again.

- For system assigned managed identities: disable and re-enable.
- For user assigned managed identities: delete, re-create, and attach them again to the necessary resources (for example, virtual machines)

Can I use a managed identity to access a resource in a different directory/tenant?

No, managed identities don't currently support cross-directory scenarios.

Are there any rate limits that apply to managed identities?

Managed identities limits have dependencies on Azure service limits, Azure Instance Metadata Service (IMDS) limits, and Microsoft Entra service limits.

- **Azure service limits** define the number of create operations that can be performed at the tenant and subscription levels. User assigned managed identities also have limitations around how they may be named.
- **IMDS** In general, requests to IMDS are limited to five requests per second. Requests exceeding this threshold are rejected with 429 responses. Requests to the Managed Identity category are limited to 20 requests per second and 5 concurrent

requests. You can read more at the Azure Instance Metadata Service (Windows) article.

- **Microsoft Entra service** Each managed identity counts towards the object quota limit in a Microsoft Entra tenant as described in Microsoft Entra service limits and restrictions.

Is it possible to move a user-assigned managed identity to a different resource group/subscription?

Moving a user-assigned managed identity to a different resource group isn't supported. If you need to use a managed identity in a different resource group or subscription, you would need to create a new user-assigned managed identity and assign the necessary permissions to it.

Are managed identities tokens cached?

Managed identity tokens are cached by the underlying Azure infrastructure for performance and resiliency purposes: the back-end services for managed identities maintain a cache per resource URI for around 24 hours. It can take several hours for changes to a managed identity's permissions to take effect, for example. Today, it isn't possible to force a managed identity's token to be refreshed before its expiry. For more information, see Limitation of using managed identities for authorization.

Are managed identities soft deleted?

Yes, Managed Identities are soft deleted for 30 days. You can view the soft deleted managed identity service principal, but you can't restore or permanently delete it.

What happens to tokens after a managed identity is deleted?

When a managed identity is deleted, an Azure resource that was previously associated with that identity can no longer request new tokens for that identity. Tokens that were issued before the identity was deleted will still be valid until their original expiry. Some target endpoints' authorization systems may carry out other checks in the directory for the identity, in which case the request fails as the object can't be found. However some systems, like Azure RBAC, will continue to accept requests from that token until it expires.

Next steps

- Learn how managed identities work with virtual machines

Feedback

Was this page helpful? Yes No

Provide product feedback

Known issues with managed identities for Azure resources

Article • 10/23/2023

This article discusses a couple of issues around managed identities and how to address them. Common questions about managed identities are documented in our frequently asked questions article.

VM fails to start after being moved

If you move a VM in a running state from a resource group or subscription, it continues to run during the move. However, after the move, if the VM is stopped and restarted, it fails to start. This issue happens because the VM doesn't update the managed identity reference and it continues to use an outdated URI.

Workaround

Trigger an update on the VM so it can get correct values for the managed identities for Azure resources. You can do a VM property change to update the reference to the managed identities for Azure resources identity. For example, you can set a new tag value on the VM with the following command:

Azure CLI

```
az vm update -n <VM Name> -g <Resource Group> --set tags.fixVM=1
```

This command sets a new tag "fixVM" with a value of 1 on the VM.

By setting this property, the VM updates with the correct managed identities for Azure resources URI, and then you should be able to start the VM.

Once the VM is started, the tag can be removed by using following command:

Azure CLI

```
az vm update -n <VM Name> -g <Resource Group> --remove tags.fixVM
```

Transferring a subscription between Microsoft Entra directories

Managed identities don't get updated when a subscription is moved/transferred to another directory. As a result, any existent system-assigned or user-assigned managed identities will be broken.

Workaround for managed identities in a subscription that has been moved to another directory:

- For system assigned managed identities: disable and re-enable.
- For user assigned managed identities: delete, re-create, and attach them again to the necessary resources (for example, virtual machines)

For more information, see Transfer an Azure subscription to a different Microsoft Entra directory.

Error during managed identity assignment operations

In rare cases, you may see error messages indicating errors related to assignment of managed identities with Azure resources. Some of the example error messages are as follows:

- Azure resource 'azure-resource-id' does not have access to identity 'managed-identity-id'.
- No managed service identities are associated with resource 'azure-resource-id'

Workaround In these rare cases the best next steps are

1. For identities no longer needed to be assigned to the resource, remove them from the resource.
2. For User Assigned Managed Identity, reassign the identity to the Azure resource.
3. For System Assigned Managed Identity, disable the identity and enable it again.

> ⓘ **Note**
>
> To assign/unassign Managed identities please follow below links

- Documentation for VM
- Documentation for VMSS

Next steps

You can review our article listing the services that support managed identities and our frequently asked questions

Azure services and resource types supporting managed identities

Article • 05/09/2025

Managed identities for Azure resources provide an automatically managed identity in Microsoft Entra ID, enabling secure, credential-free authentication to Azure services. This article lists Azure services and resource types that support managed identities.

This page provides links to services' content that can use managed identities to access other Azure resources as well as a list of Azure resource providers and resource types that support managed identities.

Additional resource provider namespace information is available in Resource providers for Azure services.

> **Important**
>
> New technical content is added daily. This list does not include every article that talks about managed identities. Please refer to each service's content set for details on their managed identities support.

Services supporting managed identities

The following Azure services support managed identities for Azure resources:

Expand table

Service Name	Documentation
API Management	Use managed identities in Azure API Management
Application Gateway	TLS termination with Key Vault certificates
Azure App Configuration	How to use managed identities for Azure App Configuration
Azure App Services	How to use managed identities for App Service and Azure Functions
Azure Arc enabled Kubernetes	Quickstart: Connect an existing Kubernetes cluster to Azure Arc
Azure Arc enabled servers	Authenticate against Azure resources with Azure Arc-enabled servers
Azure Automanage	Repair an Automanage Account

Service Name	Documentation
Azure Automation	Azure Automation account authentication overview
Azure Batch	Configure customer-managed keys for your Azure Batch account with Azure Key Vault and Managed Identity Configure managed identities in Batch pools
Azure Blueprints	Stages of a blueprint deployment
Azure Cache for Redis	Managed identity for storage accounts with Azure Cache for Redis
Azure Chaos Studio	Permissions and security in Azure Chaos Studio
Azure Communications Gateway	Deploy Azure Communications Gateway
Azure Communication Services	How to use Managed Identity with Azure Communication Services
Azure Container Apps	Managed identities in Azure Container Apps
Azure Container Instance	How to use managed identities with Azure Container Instances
Azure Container Registry	Use an Azure-managed identity in ACR Tasks
Azure CycleCloud	Using Managed Identities
Azure AI services	Configure customer-managed keys with Azure Key Vault for Azure AI services
Azure Data Box	Use customer-managed keys in Azure Key Vault for Azure Data Box
Azure Data Explorer	Configure managed identities for your Azure Data Explorer cluster
Azure Data Factory	Managed identity for Data Factory
Azure Data Lake Storage Gen1	Customer-managed keys for Azure Storage encryption
Azure Data Share	Roles and requirements for Azure Data Share
Azure DevTest Labs	Enable user-assigned managed identities on lab virtual machines in Azure DevTest Labs
Azure Digital Twins	Enable a managed identity for routing Azure Digital Twins events
Azure Event Grid	Event delivery with a managed identity
Azure Event Hubs	Authenticate a managed identity with Microsoft Entra ID to access Event Hubs Resources
Azure File Sync	How to use managed identities with Azure File Sync
Azure Files	Access Azure file shares using Microsoft Entra ID with Azure Files OAuth over REST

Service Name	Documentation
Azure Health Data Services workspace services	Authentication and authorization for Azure Health Data Services
Azure Health Data Services de-identification service	Use managed identities with the de-identification service
Azure Image Builder	Azure Image Builder overview
Azure Import/Export	Use customer-managed keys in Azure Key Vault for Import/Export service
Azure IoT Hub	IoT Hub support for virtual networks with Private Link and Managed Identity
Azure Kubernetes Service (AKS)	Use managed identities in Azure Kubernetes Service
Azure Load Testing	Use managed identities for Azure Load Testing
Azure Logic Apps	Authenticate access to Azure resources using managed identities in Azure Logic Apps
Azure Log Analytics cluster	Azure Monitor customer-managed key
Azure Machine Learning Services	Use Managed identities with Azure Machine Learning
Azure Managed Disk	Use the Azure portal to enable server-side encryption with customer-managed keys for managed disks
Azure Media services	Managed identities
Azure Monitor	Azure Monitor customer-managed key
Azure Policy	Remediate non-compliant resources with Azure Policy
Microsoft Purview	Credentials for source authentication in Microsoft Purview
Azure Quantum	Authenticate using a managed identity
Azure Resource Mover	Move resources across regions (from resource group)
Azure Site Recovery	Replicate machines with private endpoints
Azure Search	Set up an indexer connection to a data source using a managed identity
Azure Service Bus	Authenticate a managed identity with Microsoft Entra ID to access Azure Service Bus resources
Azure Service Fabric	Using Managed identities for Azure with Service Fabric
Azure SignalR Service	Managed identities for Azure SignalR Service

Service Name	Documentation
Azure Spring Apps	Enable system-assigned managed identity for an application in Azure Spring Apps
Azure SQL	Managed identities in Microsoft Entra for Azure SQL
Azure SQL Managed Instance	Managed identities in Microsoft Entra for Azure SQL
Azure Stack Edge	Manage Azure Stack Edge secrets using Azure Key Vault
Azure Static Web Apps	Securing authentication secrets in Azure Key Vault
Azure Stream Analytics	Authenticate Stream Analytics to Azure Data Lake Storage Gen1 using managed identities
Azure Synapse	Azure Synapse workspace managed identity
Azure VM image builder	Configure Azure Image Builder Service permissions using Azure CLI
Azure Virtual Machine Scale Sets	Configure managed identities on virtual machine scale set - Azure CLI
Azure Virtual Machines	Secure and use policies on virtual machines in Azure
Azure Web PubSub Service	Managed identities for Azure Web PubSub Service

Resource providers and resource types supporting managed identities

The following resource providers and resource types support managed identities:

Expand table

Namespace	ResourceType	Identity types(s)
Microsoft.AVS	privateClouds	System-assigned User-assigned
Microsoft.ApiManagement	service	System-assigned User-assigned
Microsoft.App	builders	System-assigned User-assigned
Microsoft.App	containerApps	System-assigned User-assigned

Namespace	ResourceType	Identity types(s)
Microsoft.App	jobs	System-assigned User-assigned
Microsoft.App	managedEnvironments	System-assigned User-assigned
Microsoft.App	sessionPools	System-assigned User-assigned
Microsoft.AppConfiguration	configurationStores	System-assigned User-assigned
Microsoft.AppPlatform	Spring	System-assigned
Microsoft.AppPlatform	Spring/apps	System-assigned User-assigned
Microsoft.Automation	automationAccounts	System-assigned User-assigned
Microsoft.AzureStackHCI	clusters	System-assigned
Microsoft.AzureStackHCI	devicePools	System-assigned
Microsoft.AzureStackHCI	edgeMachines	System-assigned
Microsoft.AzureStackHCI	virtualMachines	System-assigned
Microsoft.Batch	batchAccounts	System-assigned User-assigned
Microsoft.Batch	batchAccounts/pools	User-assigned
Microsoft.Blueprint	blueprintAssignments	System-assigned User-assigned
Microsoft.Cache	Redis	System-assigned User-assigned
Microsoft.Cache	redisEnterprise	System-assigned User-assigned
Microsoft.Cdn	profiles	System-assigned User-assigned
Microsoft.ChangeAnalysis	profile	System-assigned
Microsoft.CognitiveServices	accounts	System-assigned User-assigned

Namespace	ResourceType	Identity types(s)
Microsoft.CognitiveServices	accounts/encryptionScopes	
Microsoft.Communication	CommunicationServices	System-assigned User-assigned
Microsoft.Compute	diskEncryptionSets	System-assigned User-assigned
Microsoft.Compute	galleries	System-assigned User-assigned
Microsoft.Compute	virtualMachineScaleSets	System-assigned User-assigned
Microsoft.Compute	virtualMachines	System-assigned User-assigned
Microsoft.ContainerInstance	containerGroups	System-assigned User-assigned
Microsoft.ContainerInstance	containerScaleSets	System-assigned User-assigned
Microsoft.ContainerInstance	nGroups	System-assigned User-assigned
Microsoft.ContainerRegistry	registries	System-assigned User-assigned
Microsoft.ContainerRegistry	registries/credentialSets	System-assigned
Microsoft.ContainerRegistry	registries/exportPipelines	System-assigned User-assigned
Microsoft.ContainerRegistry	registries/importPipelines	System-assigned User-assigned
Microsoft.ContainerRegistry	registries/taskRuns	User-assigned
Microsoft.ContainerRegistry	registries/tasks	System-assigned User-assigned
Microsoft.ContainerService	fleets	System-assigned User-assigned
Microsoft.ContainerService	managedClusters	System-assigned User-assigned
Microsoft.ContainerService	managedclustersnapshots	System-assigned User-assigned

Namespace	ResourceType	Identity types(s)
Microsoft.ContainerService	snapshots	System-assigned User-assigned
Microsoft.CustomProviders	resourceProviders	System-assigned
Microsoft.DBforMariaDB	servers	System-assigned
Microsoft.DBforMySQL	flexibleServers	User-assigned
Microsoft.DBforMySQL	servers	System-assigned
Microsoft.DBforPostgreSQL	flexibleServers	System-assigned User-assigned
Microsoft.DBforPostgreSQL	serverGroupsv2	User-assigned
Microsoft.DBforPostgreSQL	servers	System-assigned
Microsoft.DataBox	jobs	System-assigned User-assigned
Microsoft.DataBoxEdge	DataBoxEdgeDevices	System-assigned
Microsoft.DataFactory	factories	System-assigned User-assigned
Microsoft.DataLakeStore	accounts	System-assigned
Microsoft.DataMigration	SqlMigrationServices	System-assigned
Microsoft.DataMigration	migrationServices	System-assigned
Microsoft.DataProtection	BackupVaults	System-assigned User-assigned
Microsoft.DataShare	accounts	System-assigned
Microsoft.Databricks	accessConnectors	System-assigned User-assigned
Microsoft.DesktopVirtualization	hostpools	System-assigned User-assigned
Microsoft.DevCenter	devcenters	System-assigned User-assigned
Microsoft.DevCenter	devcenters/encryptionsets	System-assigned User-assigned
Microsoft.DevCenter	projects	System-assigned User-assigned

Namespace	ResourceType	Identity types(s)
Microsoft.DevCenter	projects/environmentTypes	System-assigned User-assigned
Microsoft.DevOpsInfrastructure	pools	User-assigned
Microsoft.DevTestLab	labs	System-assigned User-assigned
Microsoft.DevTestLab	labs/serviceRunners	System-assigned User-assigned
Microsoft.DeviceUpdate	accounts	System-assigned User-assigned
Microsoft.DeviceUpdate	updateAccounts	System-assigned User-assigned
Microsoft.Devices	IotHubs	System-assigned User-assigned
Microsoft.Devices	ProvisioningServices	System-assigned User-assigned
Microsoft.DigitalTwins	digitalTwinsInstances	System-assigned User-assigned
Microsoft.DocumentDB	cassandraClusters	System-assigned
Microsoft.DocumentDB	databaseAccounts	System-assigned User-assigned
Microsoft.DocumentDB	databaseAccounts/encryptionScopes	User-assigned
Microsoft.DocumentDB	garnetClusters	System-assigned
Microsoft.DocumentDB	managedResources	System-assigned
Microsoft.DocumentDB	throughputPools	System-assigned
Microsoft.DocumentDB	throughputPools/throughputPoolAccounts	System-assigned
Microsoft.ElasticSan	elasticSans/volumeGroups	System-assigned User-assigned
Microsoft.EventGrid	domains	System-assigned User-assigned
Microsoft.EventGrid	namespaces	System-assigned User-assigned

Namespace	ResourceType	Identity types(s)
Microsoft.EventGrid	partnerTopics	System-assigned User-assigned
Microsoft.EventGrid	systemTopics	System-assigned User-assigned
Microsoft.EventGrid	topics	System-assigned User-assigned
Microsoft.EventHub	namespaces	System-assigned User-assigned
Microsoft.HDInsight	clusters	System-assigned User-assigned
Microsoft.HybridCompute	machines	System-assigned
Microsoft.HybridNetwork	networkfunctions	System-assigned User-assigned
Microsoft.HybridNetwork	publishers	System-assigned
Microsoft.HybridNetwork	serviceManagementContainers	System-assigned User-assigned
Microsoft.HybridNetwork	siteNetworkServices	System-assigned User-assigned
Microsoft.IoTCentral	IoTApps	System-assigned
Microsoft.KeyVault	managedHSMs	User-assigned
Microsoft.Kubernetes	connectedClusters	System-assigned
Microsoft.KubernetesConfiguration	extensions	System-assigned
Microsoft.Kusto	clusters	System-assigned User-assigned
Microsoft.LoadTestService	loadtests	System-assigned User-assigned
Microsoft.Logic	integrationAccounts	System-assigned User-assigned
Microsoft.Logic	integrationServiceEnvironments	System-assigned User-assigned
Microsoft.Logic	workflows	System-assigned User-assigned

Namespace	ResourceType	Identity types(s)
Microsoft.MachineLearningServices	registries	System-assigned User-assigned
Microsoft.MachineLearningServices	workspaces	System-assigned User-assigned
Microsoft.MachineLearningServices	workspaces/batchEndpoints	System-assigned
Microsoft.MachineLearningServices	workspaces/computes	System-assigned User-assigned
Microsoft.MachineLearningServices	workspaces/inferencePools/groups	System-assigned User-assigned
Microsoft.MachineLearningServices	workspaces/linkedServices	System-assigned
Microsoft.MachineLearningServices	workspaces/onlineEndpoints	System-assigned User-assigned
Microsoft.Maps	accounts	System-assigned User-assigned
Microsoft.Media	mediaservices	System-assigned User-assigned
Microsoft.Migrate	migrateprojects	System-assigned
Microsoft.Migrate	modernizeProjects	System-assigned
Microsoft.Migrate	moveCollections	System-assigned
Microsoft.MobileNetwork	mobileNetworks	User-assigned
Microsoft.MobileNetwork	packetCoreControlPlanes	User-assigned
Microsoft.MobileNetwork	simGroups	System-assigned User-assigned
Microsoft.NetApp	netAppAccounts	System-assigned User-assigned
Microsoft.Network	networkWatchers/flowLogs	User-assigned
Microsoft.OperationalInsights	clusters	System-assigned User-assigned
Microsoft.OperationalInsights	workspaces	System-assigned User-assigned
Microsoft.PowerPlatform	enterprisePolicies	System-assigned

Namespace	ResourceType	Identity types(s)
		User-assigned
Microsoft.Purview	accounts	System-assigned User-assigned
Microsoft.Quantum	Workspaces	System-assigned
Microsoft.RecoveryServices	vaults	System-assigned User-assigned
Microsoft.RedHatOpenShift	OpenShiftClusters	System-assigned
Microsoft.Search	searchServices	System-assigned User-assigned
Microsoft.Security	dataScanners	System-assigned
Microsoft.Security	pricings/securityOperators	System-assigned
Microsoft.ServiceBus	namespaces	System-assigned User-assigned
Microsoft.ServiceFabric	clusters	System-assigned User-assigned
Microsoft.ServiceFabric	clusters/applications	System-assigned User-assigned
Microsoft.ServiceFabric	managedclusters	System-assigned User-assigned
Microsoft.ServiceFabric	managedclusters/applications	System-assigned User-assigned
Microsoft.SignalRService	SignalR	System-assigned User-assigned
Microsoft.SignalRService	WebPubSub	System-assigned User-assigned
Microsoft.Solutions	applications	System-assigned User-assigned
Microsoft.Sql	managedInstances	System-assigned User-assigned
Microsoft.Sql	servers	System-assigned User-assigned
Microsoft.Sql	servers/databases	User-assigned

Namespace	ResourceType	Identity types(s)
Microsoft.Sql	servers/jobAgents	User-assigned
Microsoft.Storage	storageAccounts	System-assigned User-assigned
Microsoft.Storage	storageTasks	System-assigned
Microsoft.StorageCache	amlFilesystems	User-assigned
Microsoft.StorageCache	caches	System-assigned User-assigned
Microsoft.StorageSync	storageSyncServices	System-assigned User-assigned
Microsoft.StreamAnalytics	streamingjobs	System-assigned User-assigned
Microsoft.Synapse	workspaces	System-assigned User-assigned
Microsoft.VirtualMachineImages	imageTemplates	User-assigned
Microsoft.Web	hostingEnvironments	System-assigned User-assigned
Microsoft.Web	sites	System-assigned User-assigned
Microsoft.Web	sites/slots	System-assigned User-assigned
Microsoft.Web	staticSites	System-assigned User-assigned

Next steps

- Managed identities overview