How to use managed identities for App Service and Azure Functions

03/04/2020 • 12 minutes to read • +5

In this article

ⓘ **Important**

Managed identities for App Service and Azure Functions will not behave as expected if your app is migrated across subscriptions/tenants. The app will need to obtain a new identity, which can be done by disabling and re-enabling the feature. See Removing an identity below. Downstream resources will also need to have access policies updated to use the new identity.

This topic shows you how to create a managed identity for App Service and Azure Functions applications and how to use it to access other resources. A managed identity from Azure Active Directory (AAD) allows your app to easily access other AAD-protected resources such as Azure Key Vault. The identity is managed by the Azure platform and does not require you to provision or rotate any secrets. For more about managed identities in AAD, see Managed identities for Azure resources.

Your application can be granted two types of identities:

- A **system-assigned identity** is tied to your application and is deleted if your app is deleted. An app can only have 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.

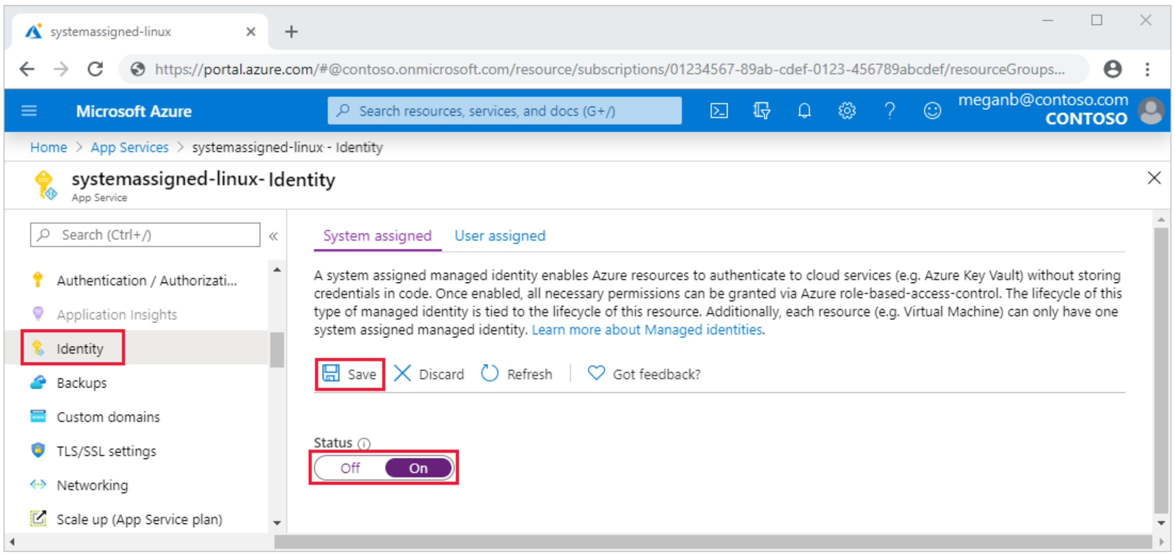
Add a system-assigned identity

Creating an app with a system-assigned identity requires an additional property to be set on the application.

Using the Azure portal

To set up a managed identity in the portal, you will first create an application as normal and then enable the feature.

1. Create an app in the portal as you normally would. Navigate to it in the portal.
2. If using a function app, navigate to **Platform features**. For other app types, scroll down to the **Settings** group in the left navigation.
3. Select **Identity**.
4. Within the **System assigned** tab, switch **Status** to **On**. Click **Save**.


systemassigned-linux
https://portal.azure.com/#@contoso.onmicrosoft.com/resource/subscriptions/01234567-89ab-cdef-0123-456789abcdef/resourceGroups...
Microsoft Azure
Search resources, services, and docs (G+/)
meganb@contoso.com
CONTOSO
Home > App Services > systemassigned-linux - Identity
systemassigned-linux- Identity
App Service
Search (Ctrl+/)
Authentication / Authorizati...
Application Insights
Identity
Backups
Custom domains
TLS/SSL settings
Networking
Scale up (App Service plan)
System assigned
User assigned
Save
Discard
Refresh
Got feedback?

A system assigned managed identity enables Azure resources to authenticate to cloud services (e.g. Azure Key Vault) without storing credentials in code. Once enabled, all necessary permissions can be granted via Azure role-based-access-control. The lifecycle of this type of managed identity is tied to the lifecycle of this resource. Additionally, each resource (e.g. Virtual Machine) can only have one system assigned managed identity. Learn more about Managed identities.

Status: Off [✓ On]

Using the Azure CLI

To set up a managed identity using the Azure CLI, you will need to use the `az webapp identity assign` command against an existing application. You have three options for running the examples in this section:

- Use Azure Cloud Shell from the Azure portal.
- Use the embedded Azure Cloud Shell via the "Try It" button, located in the top-right corner of each code block below.
- Install the latest version of Azure CLI (2.0.31 or later) if you prefer to use a local CLI console.

The following steps will walk you through creating a web app and assigning it an identity using the CLI:

1. If you're using the Azure CLI in a local console, first sign in to Azure using az login. Use an account that's associated with the Azure subscription under which you would like to deploy the application:

   Azure CLI

   ```
   az login
   ```

2. Create a web application using the CLI. For more examples of how to use the CLI with App Service, see App Service CLI samples:

   Azure CLI

   ```
   az group create --name myResourceGroup --location westus
   az appservice plan create --name myPlan --resource-group
   myResourceGroup --sku S1
   az webapp create --name myApp --resource-group myResourceGroup --plan
   myPlan
   ```

3. Run the `identity assign` command to create the identity for this application:

   Azure CLI

   ```
   az webapp identity assign --name myApp --resource-group myResourceGroup
   ```

Using Azure PowerShell

Note

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see **Introducing the new Azure PowerShell Az module**. For Az module installation instructions, see **Install Azure PowerShell**.

The following steps will walk you through creating a web app and assigning it an identity using Azure PowerShell:

1. If needed, install the Azure PowerShell using the instructions found in the Azure PowerShell guide, and then run `Login-AzAccount` to create a connection with Azure.

2. Create a web application using Azure PowerShell. For more examples of how to use Azure PowerShell with App Service, see App Service PowerShell samples:

   Azure PowerShell

   ```
   # Create a resource group.
   New-AzResourceGroup -Name myResourceGroup -Location $location

   # Create an App Service plan in Free tier.
   New-AzAppServicePlan -Name $webappname -Location $location -
   ResourceGroupName myResourceGroup -Tier Free

   # Create a web app.
   New-AzWebApp -Name $webappname -Location $location -AppServicePlan
   $webappname -ResourceGroupName myResourceGroup
   ```

3. Run the `Set-AzWebApp -AssignIdentity` command to create the identity for this application:

   Azure PowerShell

   ```
   Set-AzWebApp -AssignIdentity $true -Name $webappname -ResourceGroupName
   myResourceGroup
   ```

Using an Azure Resource Manager template

An Azure Resource Manager template can be used to automate deployment of your Azure resources. To learn more about deploying to App Service and Functions, see Automating resource deployment in App Service and Automating resource deployment in Azure Functions.

Any resource of type `Microsoft.Web/sites` can be created with an identity by including the following property in the resource definition:

JSON

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

Note

An application can have both system-assigned and user-assigned identities at the same time. In this case, the `type` property would be `SystemAssigned,UserAssigned`

Adding the system-assigned type tells Azure to create and manage the identity for your application.

For example, a web app might look like the following:

JSON

Copy

```json
{
    "apiVersion": "2016-08-01",
    "type": "Microsoft.Web/sites",
    "name": "[variables('appName')]",
    "location": "[resourceGroup().location]",
    "identity": {
        "type": "SystemAssigned"
    },
    "properties": {
        "name": "[variables('appName')]",
        "serverFarmId": "[resourceId('Microsoft.Web/serverfarms',
variables('hostingPlanName'))]",
        "hostingEnvironment": "",
        "clientAffinityEnabled": false,
        "alwaysOn": true
    },
    "dependsOn": [
        "[resourceId('Microsoft.Web/serverfarms',
variables('hostingPlanName'))]"
    ]
}
```

When the site is created, it has the following additional properties:

JSON

Copy

```json
"identity": {
    "type": "SystemAssigned",
    "tenantId": "<TENANTID>",
    "principalId": "<PRINCIPALID>"
}
```

The tenantId property identifies what AAD tenant the identity belongs to. The principalId is a unique identifier for the application's new identity. Within AAD, the service principal has the same name that you gave to your App Service or Azure Functions instance.

Add a user-assigned identity

Creating an app with a user-assigned identity requires that you create the identity and then add its resource identifier to your app config.

Using the Azure portal

First, you'll need to create a user-assigned identity resource.

1. Create a user-assigned managed identity resource according to these instructions.

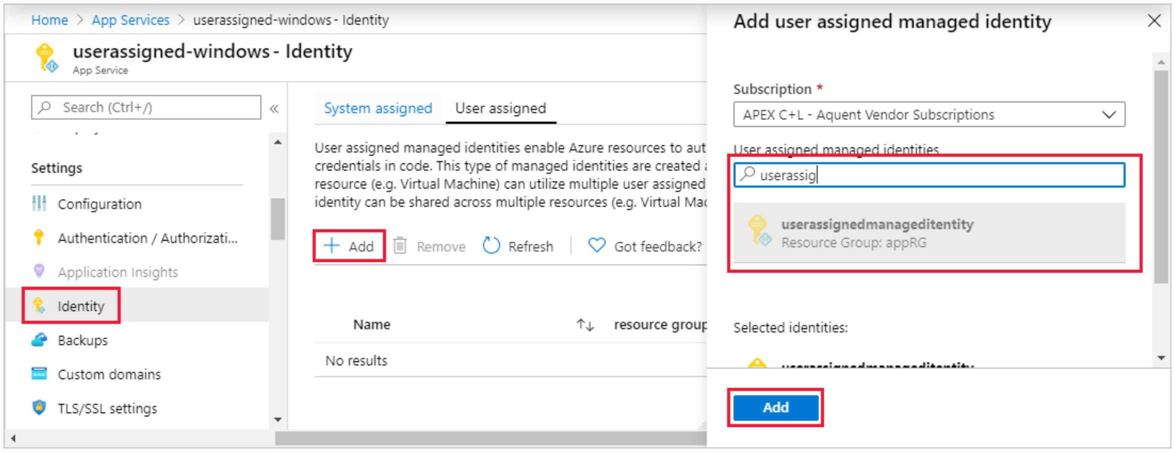
2. Create an app in the portal as you normally would. Navigate to it in the portal.

3. If using a function app, navigate to **Platform features**. For other app types, scroll down to the **Settings** group in the left navigation.

4. Select **Identity**.

5. Within the **User assigned** tab, click **Add**.

6. Search for the identity you created earlier and select it. Click **Add**.



User assigned managed identities enable Azure resources to aut
credentials in code. This type of managed identities are created
resource (e.g. Virtual Machine) can utilize multiple user assigned
identity can be shared across multiple resources (e.g. Virtual Ma

Subscription *
APEX C+L - Aquent Vendor Subscriptions

User assigned managed identities
userassig

userassignedmanageditentity
Resource Group: appRG

Selected identities:

Using an Azure Resource Manager template

An Azure Resource Manager template can be used to automate deployment of your Azure resources. To learn more about deploying to App Service and Functions, see Automating resource deployment in App Service and Automating resource deployment in Azure Functions.

Any resource of type `Microsoft.Web/sites` can be created with an identity by including the following block in the resource definition, replacing `<RESOURCEID>` with the resource ID of the desired identity:

JSON

Copy

```json
"identity": {
    "type": "UserAssigned",
    "userAssignedIdentities": {
        "<RESOURCEID>": {}
    }
}
```

Note

An application can have both system-assigned and user-assigned identities at the same time. In this case, the `type` property would be `SystemAssigned,UserAssigned`

Adding the user-assigned type tells Azure to use the user-assigned identity specified for your application.

For example, a web app might look like the following:

JSON Copy

```json
{
    "apiVersion": "2016-08-01",
    "type": "Microsoft.Web/sites",
    "name": "[variables('appName')]",
    "location": "[resourceGroup().location]",
    "identity": {
        "type": "UserAssigned",
        "userAssignedIdentities": {
            "[resourceId('Microsoft.ManagedIdentity/userAssignedIdentities',
variables('identityName'))]": {}
        }
    },
    "properties": {
        "name": "[variables('appName')]",
        "serverFarmId": "[resourceId('Microsoft.Web/serverfarms',
variables('hostingPlanName'))]",
        "hostingEnvironment": "",
        "clientAffinityEnabled": false,
        "alwaysOn": true
    },
    "dependsOn": [
        "[resourceId('Microsoft.Web/serverfarms',
variables('hostingPlanName'))]",
        "[resourceId('Microsoft.ManagedIdentity/userAssignedIdentities',
variables('identityName'))]"
    ]
}
```

When the site is created, it has the following additional properties:

JSON Copy

```json
"identity": {
    "type": "UserAssigned",
    "userAssignedIdentities": {
        "<RESOURCEID>": {
            "principalId": "<PRINCIPALID>",
            "clientId": "<CLIENTID>"
        }
    }
}
```

The principalId is a unique identifier for the identity that's used for AAD administration. The clientId is a unique identifier for the application's new identity that's used for specifying which identity to use during runtime calls.

Obtain tokens for Azure resources

An app can use its managed identity to get tokens to access other resources protected by AAD, such as Azure Key Vault. These tokens represent the application accessing the resource, and not any specific user of the application.

You may need to configure the target resource to allow access from your application. For example, if you request a token to access Key Vault, you need to make sure you have added an access policy that includes your application's identity. Otherwise, your calls to Key Vault will be rejected, even if they include the token. To learn more about which resources support Azure Active Directory tokens, see Azure services that support Azure AD authentication.

Important

The back-end services for managed identities maintain a cache per resource URI for around 8 hours. If you update the access policy of a particular target resource and immediately retrieve a token for that resource, you may continue to get a cached token with outdated permissions until that token expires. There's currently no way to force a token refresh.

There is a simple REST protocol for obtaining a token in App Service and Azure Functions. This can be used for all applications and languages. For .NET and Java, the Azure SDK provides an abstraction over this protocol and facilitates a local development experience.

Using the REST protocol

An app with a managed identity has two environment variables defined:

- MSI_ENDPOINT - the URL to the local token service.
- MSI_SECRET - a header used to help mitigate server-side request forgery (SSRF) attacks. The value is rotated by the platform.

The **MSI_ENDPOINT** 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, including the following parameters:

Parameter name	In	Description
resource	Query	The AAD resource URI of the resource for which a token should be obtained. This could be one of the Azure services that support Azure AD authentication or any other resource URI.
api-version	Query	The version of the token API to be used. "2017-09-01" is currently the only version supported.
secret	Header	The value of the MSI_SECRET environment variable. This header is used to help mitigate server-side request forgery (SSRF) attacks.
clientid	Query	(Optional unless for user-assigned) The ID of the user-assigned identity to be used. If omitted, the system-assigned identity is used.

ⓘ **Important**

If you are attempting to obtain tokens for user-assigned identities, you must include the `clientid` property. Otherwise the token service will attempt to obtain a token for a system-assigned identity, which may or may not exist.

A successful 200 OK response includes a JSON body with the following properties:

Property name	Description
access_token	The requested access token. The calling web service can use this token to authenticate to the receiving web service.
expires_on	The time when the access token expires. The date is represented as the number of seconds from 1970-01-01T0:0:0Z UTC until the expiration time. This value is used to determine the lifetime of cached tokens.
resource	The App ID URI of the receiving web service.

Property name	Description
token_type	Indicates the token type value. The only type that Azure AD supports is Bearer. For more information about bearer tokens, see The OAuth 2.0 Authorization Framework: Bearer Token Usage (RFC 6750).

This response is the same as the response for the AAD service-to-service access token request.

> **Note**
>
> Environment variables are set up when the process first starts, so after enabling a managed identity for your application, you may need to restart your application, or redeploy its code, before `MSI_ENDPOINT` and `MSI_SECRET` are available to your code.

REST protocol examples

An example request might look like the following:

Copy

```
GET /MSI/token?resource=https://vault.azure.net&api-version=2017-09-01
HTTP/1.1
Host: localhost:4141
Secret: 853b9a84-5bfa-4b22-a3f3-0b9a43d9ad8a
```

And a sample response might look like the following:

Copy

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

{
    "access_token": "eyJ0eXAi…",
    "expires_on": "09/14/2017 00:00:00 PM +00:00",
    "resource": "https://vault.azure.net",
    "token_type": "Bearer"
}
```

Code examples

.NET JavaScript Python PowerShell

Tip

For .NET languages, you can also use Microsoft.Azure.Services.AppAuthentication instead of crafting this request yourself.

C# Copy

```
private readonly HttpClient _client;
// ...
public async Task<HttpResponseMessage> GetToken(string resource)  {
    var request = new HttpRequestMessage(HttpMethod.Get,
        String.Format("{0}/?resource={1}&api-version=2017-09-01",
Environment.GetEnvironmentVariable("MSI_ENDPOINT"), resource));
    request.Headers.Add("Secret",
Environment.GetEnvironmentVariable("MSI_SECRET"));
    return await _client.SendAsync(request);
}
```

Using the Microsoft.Azure.Services.AppAuthentication library for .NET

For .NET applications and functions, the simplest way to work with a managed identity is through the Microsoft.Azure.Services.AppAuthentication package. This library will also allow you to test your code locally on your development machine, 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 any other necessary NuGet packages to your application. The below example also uses Microsoft.Azure.KeyVault.

2. Add the following code to your application, modifying to target the correct resource. This example shows two ways to work with Azure Key Vault:

   C# Copy

   ```
   using Microsoft.Azure.Services.AppAuthentication;
   using Microsoft.Azure.KeyVault;
   // ...
   var azureServiceTokenProvider = new AzureServiceTokenProvider();
   ```

```
string accessToken = await
azureServiceTokenProvider.GetAccessTokenAsync("https://vault.azure.net"
);
// 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 MSI .NET sample.

Using the Azure SDK for Java

For Java applications and functions, the simplest way to work with a managed identity is through the Azure SDK for Java. This section shows you how to get started with the library in your code.

1. Add a reference to the Azure SDK library. For Maven projects, you might add this snippet to the `dependencies` section of the project's POM file:

   XML Copy

   ```xml
   <dependency>
       <groupId>com.microsoft.azure</groupId>
       <artifactId>azure</artifactId>
       <version>1.23.0</version>
   </dependency>
   ```

2. Use the `AppServiceMSICredentials` object for authentication. This example shows how this mechanism may be used for working with Azure Key Vault:

   Java Copy

   ```java
   import com.microsoft.azure.AzureEnvironment;
   import com.microsoft.azure.management.Azure;
   import com.microsoft.azure.management.keyvault.Vault
   //...
   Azure azure = Azure.authenticate(new
   AppServiceMSICredentials(AzureEnvironment.AZURE))
           .withSubscription(subscriptionId);
   Vault myKeyVault = azure.vaults().getByResourceGroup(resourceGroup,
   keyvaultName);
   ```

Remove an identity

A system-assigned identity can be removed by disabling the feature using the portal, PowerShell, or CLI in the same way that it was created. User-assigned identities can be removed individually. To remove all identities, set the type to "None" in the ARM template:

JSON Copy

```json
"identity": {
    "type": "None"
}
```

Removing a system-assigned identity in this way will also delete it from AAD. System-assigned identities are also automatically removed from AAD when the app resource is deleted.

Note

There is also an application setting that can be set, WEBSITE_DISABLE_MSI, which just disables the local token service. However, it leaves the identity in place, and tooling will still show the managed identity as "on" or "enabled." As a result, use of this setting is not recommended.

Next steps

Access SQL Database securely using a managed identity

Is this page helpful?

Yes No