



Alejandro Sanz Martín

Implementing Azure Managed Identity

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

3 May 2022

Abstract

Author: Alejandro Sanz Martín
Title: Implementing Azure Managed Identity
Number of Pages: 26 pages
Date: 3.5.2022

Degree: Bachelor of Engineering
Degree Programme: Information Technology
Professional Major: Internet of Things
Supervisors: Janne Salonen, Head of Department

The main objective of this project is to give the reader a good understanding of what azure managed identity is and to demonstrate how to implement it in a normal scenario. It also shows the code that is needed to develop an application that is able to use azure managed identity. This project will not cover the creation of Azure Active Directory, assuming that the reader already knows how to create it properly.

The project will start with an introduction of the importance of security for companies, followed by an overview of the material and methods used during the project. The general concepts related to the project and the basic scenario used during the project are also explained. After that, there will be a demonstration of how all the Azure resources used in this project were created.

The project shows two different demonstrations of how to implement Azure managed identity. The first demonstration implements a system assigned identity, while the second demonstration implements a user assigned identity.

The project ends with the main conclusion, that Azure managed identity offers a good solution to secure connections between different Azure resources. It also offers a possibility of removing all the possible passwords from the code, that would be needed for establishing a connection between different resources.

Keywords: Microsoft, Azure, Managed Identity

Contents

List of Abbreviations

1	Introduction	1
2	Material and Methods	2
3	Theoretical Background	2
3.1	What is Microsoft Azure	2
3.2	Managed Identity	3
3.2.1	System Assigned Managed Identity	4
3.2.2	User Assigned Managed Identity	6
4	Scenario	7
4.1	Overview	7
4.2	Prerequisites	8
4.3	Implementation	9
4.3.1	Web Application	9
4.3.2	Data Base	10
4.3.3	Storage Account	14
4.4	System-Assigned Managed Identity Implementation	15
4.4.1	Overview	15
4.4.2	Creating the system assigned identity	16
4.4.3	Configuring the database	17
4.4.4	Configuring the storage account	17
4.4.5	Code implementation	18
4.5	User-Assigned Managed Identity Implementation	20
4.5.1	Overview	20
4.5.2	Creating and using the user assigned identity	21
4.5.3	Configuring the database	22
4.5.4	Configuring the storage account	23
4.5.5	Code implementation	24
5	Conclusion	25

References

List of Abbreviations

ISP: Internet Service Provider. Company which provides internet connection to organizations or individuals.

IaaS: Infrastructure as a Service. Cloud computer service model type.

PaaS: Platform as a Service. Cloud computer service model type.

SaaS: Software as a Service. Cloud computer service model type.

AD: Active Directory. Identity and access management service.

1 Introduction

Currently many companies offer online services through the internet and use Microsoft Azure as a platform. Normally these companies store a big variety of sensitive information of their users. Any breach in the system or vulnerability in the code could lead to exposure of the user's private information.

Companies are responsible for protecting the information and privacy of their users. Loosing this information might have legal consequences and affect the image of the company. It might also affect their users in many different ways, depending on the type of information that is compromised. For this reason, companies spend a great amount of money, time and resources in cybersecurity. In some cases, companies hire third-party companies to ensure the security.

Clients of Microsoft Azure can benefit of a great variety of services oriented to secure their resources. Because it is offered by Microsoft, it means that the company doesn't have to worry about all the security aspects related with the resource and it can focus on the code and application security aspects. The company can save money, time and resources that the company would need to employ to get the same result.

One of the multiple services that Microsoft Azure offers to their clients, is Managed Identity. This service, which does not have any costs, provides the resources with an identity inside of Microsoft Azure Active Directory which can be used by developers to eliminate the needed credentials from the code or connection strings.

In this project, the reader will get an understanding of what Azure managed identity is and how it can be used. The project also shows how to configure the Azure resources and how to implement it in the code.

2 Material and Methods

The material used in this project is mainly books and Microsoft documentation that is available online. The books are strongly based on Azure infrastructure and authentication. They have been used to get a good base knowledge about Azure in general and the resource deployment process. The knowledge coming from these books was important during the phase of creating the scenario.

The Microsoft documentation used was focused on the main topic, managed identity and its implementation. This documentation has been the key for this project. It provided the information needed to understand, what is managed identity and its different types, as well as the classes and functions available to develop applications using these identities.

The method used was mainly trying and comparing the results, using the knowledge acquired from the material. The log files provided by the Azure resources, combined with the debugger tools of Visual Studio, were the key to detect errors and to find solutions.

3 Theoretical Background

3.1 What is Microsoft Azure

Before being published, Microsoft Azure used to be referred with the codename Red Dog (RD) ^[1]. On October 27, 2008, it was announced with the name of Windows Azure at the Professional Developers Conference, and it was available as Community Technical Preview (CTP) ^[1, 2]. In 2010 it became commercially available ^[1, 2]. Currently Windows Azure is known as Microsoft Azure ^[2].

Microsoft Azure is one of the principal cloud computing providers available in the market. Azure offers a big variety of different kinds of resources and service models like PaaS, SaaS and IaaS, hosted on top of its own infrastructure. ^[1, 2, 3]

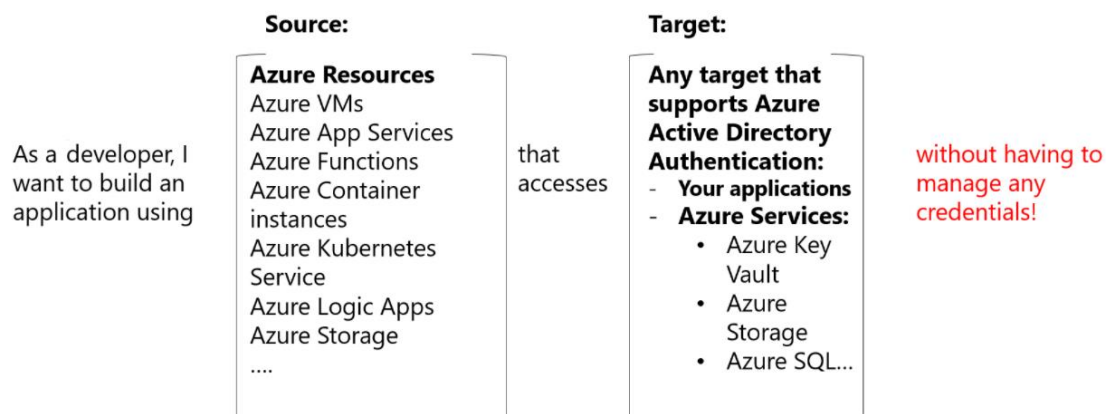
Cloud providers, such as Microsoft Azure, offer companies a cost-effective alternative to the traditional on-premises infrastructure. Companies using cloud computing do not have to consider the cost of licenses, hardware, maintenance, expansions, the space needed, the electricity or ISP. [2, 3]

3.2 Managed Identity

Managed identity is a free feature, provided by Microsoft Azure, which allows different Azure resources to authenticate and authorize themselves with other Azure resources that support this feature. [4, 5, 6, 7]

Managed identity allows code developers to escape, without additional cost, the challenge that represent the management of credentials, which are used to protect the communication between different components of the solution. Figure 1 shows this idea simplified with a small list of resources that support managed identity. [4, 5, 6,7]

I can use Managed Identities when...



For example, I want to build an application using **Azure App Services** that accesses **Azure Storage** without having to manage any credentials.

Figure 1 Basic Idea of Azure Managed Identity [4]

Managed identity, or Managed Service Identity (MSI) as it was formerly known, requires that the source and the target resource or service supports Azure AD

authentication in order to establish the connection without managing credentials.^[4]

Currently there are two different types of managed identity. The first one is System Assigned Managed Identity and the second one is User Assigned Identity. Both have the same purpose but the way of using them and their life cycle is different. ^[4, 6]

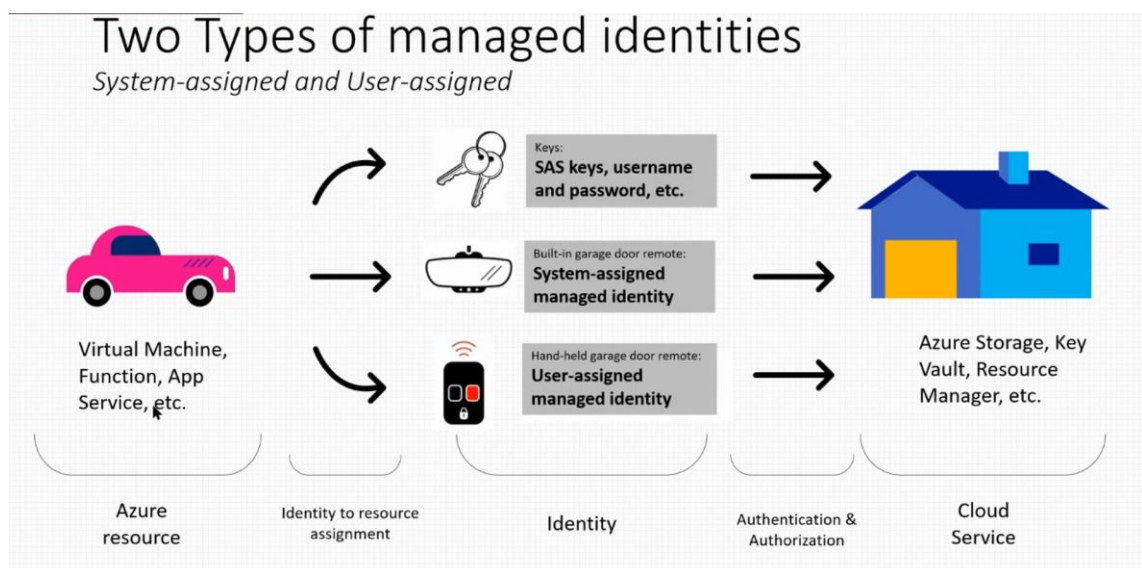


Figure 2 Representation of the different types of Managed Identity ^[4]

Figure 2 shows a simple representation of both types of managed identity, user assigned and system assigned. The system assigned managed identity is represented as a built-in garage door remote, which can only be used by one car. Same way, the system assigned identity can only be used by one resource. The user assigned managed identity is represented as a hand-held garage door remote, which can be used from many different cars. The user assigned identity can be used by one or more applications the same way.

3.2.1 System Assigned Managed Identity

System assigned identity is represented as an option that can be turned on or off for resources that support managed identity. When it is turned on, one Azure

AD identity is assigned exclusively for that resource, and it can be used to authenticate that resource in Azure.^[4]

Table 1 System-Assigned Managed Identity Properties ^[4]

Property	System-assigned managed identity
Creation	Created as part of an Azure resource (for example, an Azure virtual machine or Azure App Service)
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.
Sharing across Azure resources	Cannot be shared. It can only be associated with a single 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

It is possible to create only one system assigned managed identity for each Azure resource, and as Table 1 shows, it is bounded to the resource and its life cycle. This also means that a system assigned identity cannot be shared with other Azure resources.^[4]

System assigned managed identity is the best option in scenarios where each resource requires a specific set of permissions or where logging the specific activity of each resource is needed. Another scenario where system assigned

identity is recommended to be used is when it is required that the permissions granted to a resource are removed along with the resource. [8]

3.2.2 User Assigned Managed Identity

User assigned identity is represented as a standalone Azure resource, and it can be assigned to one or more Azure resources. This identity is independent from the associated Azure resources, which means that it is not bounded to the related resource life cycle, as Table 2 shows. This type of identity can only be deleted manually. [4, 8]

Table 2 User Assigned Managed Identity Properties [4]

Property	User-assigned managed identity
Creation	Created as a stand-alone Azure resource
Life cycle	Independent life cycle. Must be explicitly deleted.
Sharing across Azure resources	Can be shared The same user-assigned managed identity can be associated with more than one Azure resource.
Common use cases	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	User-assigned managed identity
	For example, a workload where multiple virtual machines need to access the same resource

In most scenarios, user assigned managed identity is more efficient than system assigned. For scenarios where multiple resources run the same task or access the same resources, user assigned identity is the best solution. In scenarios where multiple resources are being created or deleted in a short period of time, using system assigned identities might reach the Azure Active Directory rate limit and it would end with a HTTP 429 error. However, using user assigned identities would avoid the rate limit deploying resources associated with a single user assigned identity. ^[8]

4 Scenario

4.1 Overview

In this project there will be two scenarios created that are based on the same idea but implementing different managed identity types to show how to implement them from the beginning. Both scenarios will be based on PaaS and will use the same Azure resources.

This project focuses on the needed configuration on the Azure resources and the code changes, in order to implement and use managed identity. The project will not cover the creation of Azure Active Directory, assuming that the reader has a general understanding of how to do it.

The scenario will be based in one web application hosting a web page which will access a data base and a storage account. At the same time, the web application will record all the logs in a storage account.

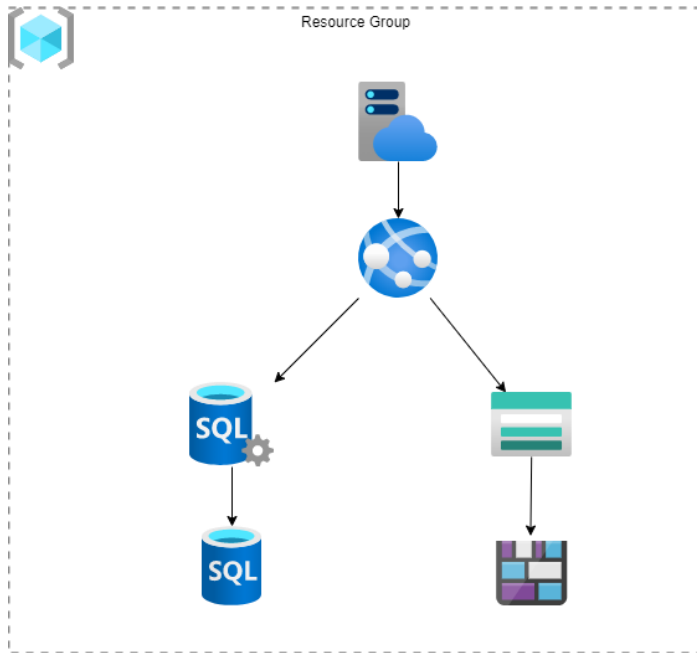


Figure 3 Basic Scenario

The picture 3 shows a representation of this basic scenario, connecting a web application with a SQL database and a storage account which contains a blob container.

4.2 Prerequisites

In order to recreate this scenario and to be able to use managed identity, it will be necessary to meet the following requirements:

- One Azure account with a subscription
- Azure Active Directory
- Resources that support managed identity

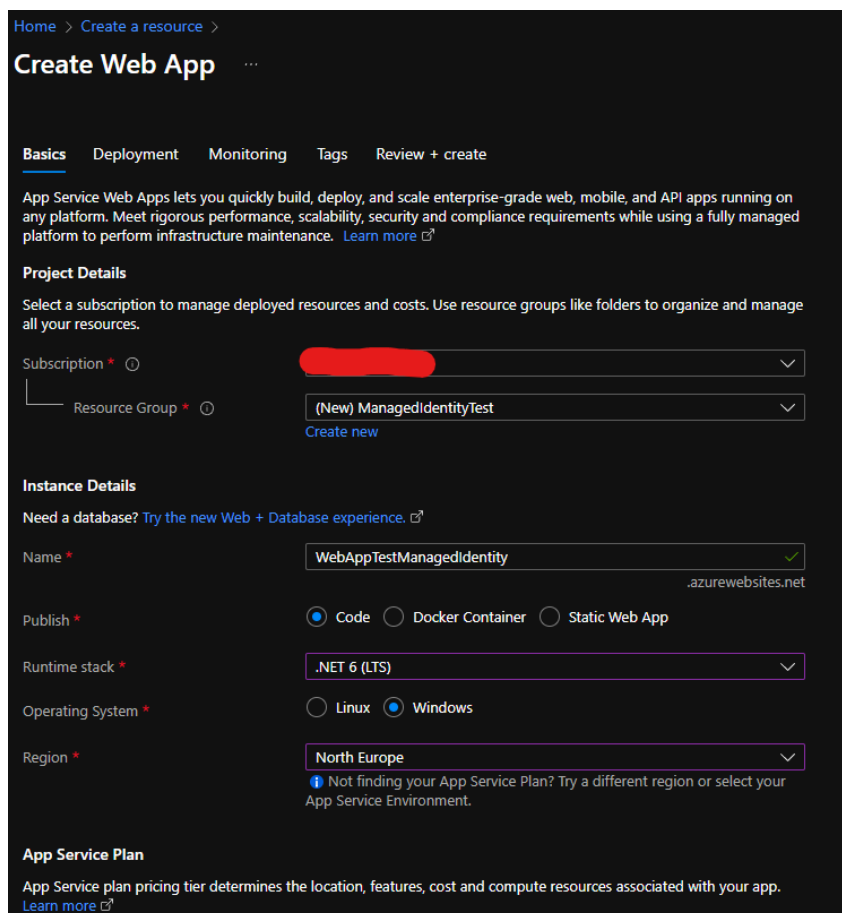
In this scenario will be used resources with minimum tier to keep the cost as low as possible.

4.3 Implementation

This section will cover the needed steps to create and configure the basic scenario that will be used to implement the different types of managed identities in following sections.

4.3.1 Web Application

Like for any Azure resource, the first step is to create it from the Azure portal home page, clicking on the “Create a resource” button. After clicking there, the next step is to find the resource that will be created, in this case, “Web App”. Figure 4 shows an example of some of the options available for the web app creation.



The screenshot shows the 'Create Web App' configuration panel in the Azure portal. The page is titled 'Create Web App' and has a breadcrumb trail 'Home > Create a resource >'. Below the title are tabs for 'Basics', 'Deployment', 'Monitoring', 'Tags', and 'Review + create'. The 'Basics' tab is selected. The main content area is divided into several sections:

- Project Details:** A description of App Service Web Apps and instructions to select a subscription and resource group. The 'Subscription' dropdown is redacted with a red oval. The 'Resource Group' dropdown is set to '(New) ManagedIdentityTest' with a 'Create new' link below it.
- Instance Details:** A section for configuring the app instance. It includes a link for 'Need a database? Try the new Web + Database experience.' and the following fields:
 - Name:** 'WebAppTestManagedIdentity' with a green checkmark and '.azurewebsites.net' suffix.
 - Publish:** Radio buttons for 'Code' (selected), 'Docker Container', and 'Static Web App'.
 - Runtime stack:** Dropdown menu set to '.NET 6 (LTS)'.
 - Operating System:** Radio buttons for 'Linux' and 'Windows' (selected).
 - Region:** Dropdown menu set to 'North Europe' with a help message: 'Not finding your App Service Plan? Try a different region or select your App Service Environment.'
- App Service Plan:** A section for selecting a pricing tier, with a 'Learn more' link.

Figure 4 Web App Creation Panel

From figure 4, all the needed parameters for the creation of the web app have been introduced. Figure 5 shows the missed parameters from the figure 4 that refers to the creation of the app service plan that will host the web app.

The screenshot shows the 'App Service Plan' configuration page. At the top, it states: 'App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. [Learn more](#)'. Below this, there are three main sections:

- Windows Plan (North Europe) ***: A dropdown menu is set to '(New) WebPlanTestManagedIdentity' with a 'Create new' link below it.
- Skus and size ***: The selected option is 'Free F1' with the description 'Shared infrastructure, 1 GB memory' and a 'Change size' link.
- Zone redundancy**: A text block explains that this is a deployment-time decision. Below it, two radio button options are shown:
 - Enabled:** Your App Service plan and the apps in it will be zone redundant. The minimum App Service plan instance count will be three.
 - Disabled:** Your App Service Plan and the apps in it will not be zone redundant. The minimum App Service plan instance count will be one.

At the bottom of the form, there are three buttons: 'Review + create' (highlighted in blue), '< Previous', and 'Next : Deployment >'.

Figure 5 App Service Plan Creation

The rest of the options from the other tabs can keep the default parameters. After reviewing the parameters and clicking on the create button, it might take a couple of minutes to create the resources.

4.3.2 Data Base

The data base can be created from the home page, clicking on the "Create a resource" button as previously. Then, the next step is to find and select the resource with a name "SQL Database". Figure 6 shows an example of some of the options available for the SQL Database creation.

Microsoft Azure Search resources, services, and docs (G+)

Home > Create a resource >

Create SQL Database

Microsoft

Basics Networking Security Additional settings Tags Review + create

Create a SQL database with your preferred configurations. Complete the Basics tab then go to Review + Create to provision with smart defaults, or visit each tab to customize. [Learn more](#)

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *

Resource group * [Create new](#)

Database details

Enter required settings for this database, including picking a logical server and configuring the compute and storage resources

Database name * ✓

Server * [Create new](#)

Want to use SQL elastic pool? * Yes No

Compute + storage * **General Purpose**
Serverless, Gen5, 1 vCore, 32 GB storage, zone redundant disabled
[Configure database](#)

Backup storage redundancy

Choose how your PITR and LTR backups are replicated. Geo restore or ability to recover from regional outage is only available when geo-redundant storage is selected.

[Review + create](#) [Next: Networking >](#)

Figure 6 SQL Data Base Creation Panel

In figure 6, the parameters needed for the creation of the data base, have already been introduced. The resource group chosen is the same as the web app uses. For this scenario, it will not be necessary to use SQL elastic pool due there is not scheduled tasks to run.

Microsoft Azure Search resources, services, and docs (G+)

Home > Create a resource > Create SQL Database >

Create SQL Database Server

Microsoft

Server details

Enter required settings for this server, including providing a name and location. This server will be created in the same subscription and resource group as your database.

Server name * .database.windows.net

Location *

Authentication

Select your preferred authentication methods for accessing this server. Create a server admin login and password to access your server with SQL authentication, select only Azure AD authentication [Learn more](#) using an existing Azure AD user, group, or application as Azure AD admin [Learn more](#), or select both SQL and Azure AD authentication.

Authentication method

- Use SQL authentication
- Use only Azure Active Directory (Azure AD) authentication
- Use both SQL and Azure AD authentication

Set Azure AD admin

Admin Object/App ID: [REDACTED]
[Set admin](#)

Server admin login *

Password *

Confirm password *

OK

Figure 7 SQL Database Server Creation Panel

SQL Database requires a SQL Server to serve it. SQL Server can be created by clicking on the “Create server” button showed in figure 6. Figure 7 shows the SQL server creation panel with the parameters that will be used in this scenario. For the SQL database, the name must be in lowercase letters. It is very important that the SQL server allows the Azure Active Directory authentication. In order to create database contained users that authenticate using Azure AD, it is necessary to be logged with an Azure AD account.

Microsoft Azure Search resources, services, and docs (G+)

Home > Create a resource > Create SQL Database >

Configure

Feedback

Service and compute tier

Select from the available tiers based on the needs of your workload. The vCore model provides a wide range of configuration controls and offers Hyperscale and Serverless to automatically scale your database based on your workload needs. Alternately, the DTU model provides set price/performance packages to choose from for easy configuration. [Learn more](#)

Service tier: **General Purpose (Scalable compute and storage options)** [Compare service tiers](#)

Compute tier:

- Provisioned** - Compute resources are pre-allocated. Billed per hour based on vCores configured.
- Serverless** - Compute resources are auto-scaled. Billed per second based on vCores used.

Compute Hardware

Select the hardware configuration based on your workload requirements. Availability of compute optimized, memory optimized, and confidential computing hardware depends on the region, service tier, and compute tier.

Hardware Configuration: **Gen5**
up to 40 vCores, up to 120 GB memory
[Change configuration](#)

Max vCores: 1

Min vCores: 0.5 vCores

2.02 GB MIN MEMORY 3 GB MAX MEMORY

Auto-pause delay

The database automatically pauses if it is inactive for the time period specified here, and automatically resumes when database activity recurs. Alternatively, auto-pausing can be disabled.

[Apply](#)

Figure 8 Database Compute and Memory Configuration Panel

Figure 8 shows the panel that appears after clicking on the “Configure database” button from figure 6, and it allows to configure the compute tier and the storage capacity. For this scenario, using the serverless compute tier is the only change applied based on reducing costs.

Backup storage redundancy

Choose how your PITR and LTR backups are replicated. Geo restore or ability to recover from regional outage is only available when geo-redundant storage is selected.

Backup storage redundancy:

- Locally-redundant backup storage**
- Zone-redundant backup storage
- Geo-redundant backup storage

[Review + create](#) [Next: Networking >](#)

Figure 9 Backup Storage Redundancy

Figure 9 shows the backup storage redundancy section that did not fit in figure 6. For this scenario, there is no need for a redundancy solution. The option chosen is the “Locally-redundant backup storage”. The rest of the options in the other tags can stay as default.

4.3.3 Storage Account

The storage account can be created, as the previous, from the home page by clicking on the “Create a resource” button and choosing the resource “Storage account”. Figure 10 shows the main option available for the storage account creation.

Home > Create a resource >

Create a storage account

Basics | Advanced | Networking | Data protection | Encryption | Tags | Review + create

Azure Storage is a Microsoft-managed service providing cloud storage that is highly available, secure, durable, scalable, and redundant. Azure Storage includes Azure Blobs (objects), Azure Data Lake Storage Gen2, Azure Files, Azure Queues, and Azure Tables. The cost of your storage account depends on the usage and the options you choose below. [Learn more about Azure storage accounts](#)

Project details

Select the subscription in which to create the new storage account. Choose a new or existing resource group to organize and manage your storage account together with other resources.

Subscription *

Resource group * [Create new](#)

Instance details

If you need to create a legacy storage account type, please click [here](#).

Storage account name ⓘ *

Region ⓘ *

Performance ⓘ *

- Standard: Recommended for most scenarios (general-purpose v2 account)
- Premium: Recommended for scenarios that require low latency.

Redundancy ⓘ *

[Review + create](#) [< Previous](#) [Next : Advanced >](#)

Figure 10 Storage Account Creation Panel

For this scenario the parameters that need to be configured are shown in figure 10. The storage account name must be in lowercase letters. Once again, there is no need for redundancy in this scenario, so the option chosen is the “Locally-redundant storage”. The rest of the parameters in other tabs can stay as default.

Once the storage account is created, it is time to create the blob container which will store the log files from the web application for this scenario. This can be done from the “Containers” section in the storage account menu, clicking on the “+ Container” button.

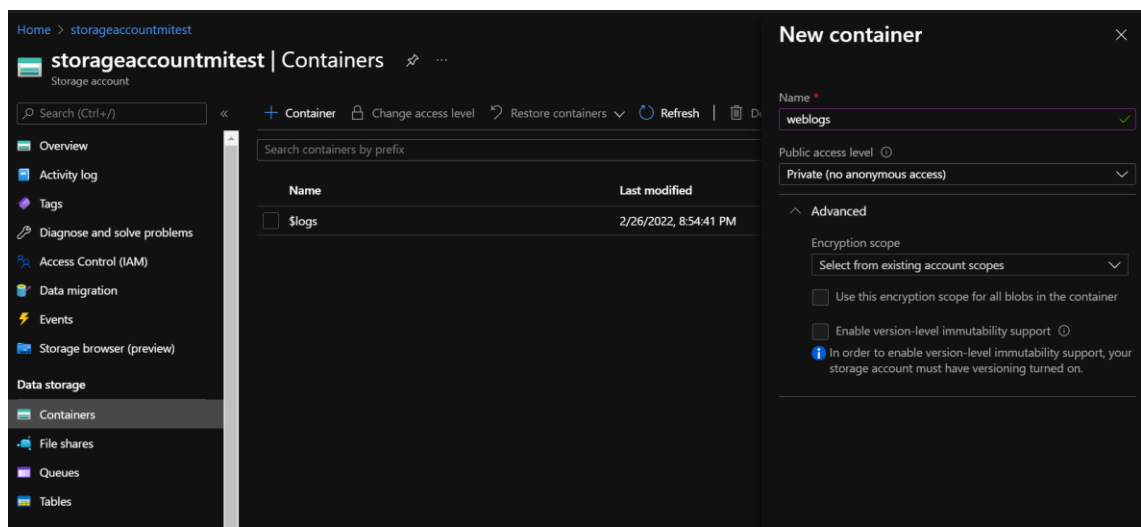


Figure 11 Blob Container Creation Panel

The process of creating a new blob container is shown in figure 11. The name must be in lowercase letters, and for this scenario, the access level will be set as private, and the encryption options will stay as default.

4.4 System-Assigned Managed Identity Implementation

4.4.1 Overview

In this scenario, a system assigned managed identity will be generated. The identity will be bounded to the web application, which will use it to authenticate with the database and the storage account.

A new database contained user will be created for the database, which will be declared as an Azure AD user. This means that the database will not store a password for this user. The user needs to have the same name as the web application, and it will represent an Azure AD user and Azure will handle the authentication. Once the user is created, the needed permissions have to be granted to it.

In case of the storage account, it is only necessary to grant permissions to the web application. The web application is treated as a normal Azure AD user after creating the managed identity.

4.4.2 Creating the system assigned identity

The system assigned identity is created from the Azure resource which will be associated with, in this case, the web application. The process for generating this identity is very simple. The only step needed, is to switch the status option under the system assigned section, from the “Identity” setting in the web application menu.

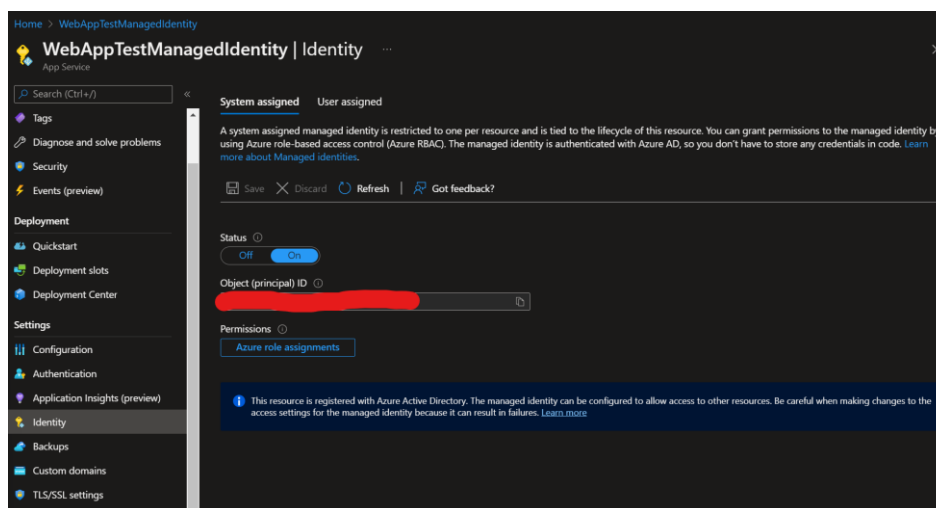


Figure 12 System Assigned Managed Identity Panel

Figure 12 shows the managed identity panel of the web application. The system assigned managed identity has been enabled and one new identity has been generated and associated with the web application. At this point the web application can use this identity to authenticate with any Azure resource which supports managed identity.

4.4.3 Configuring the database

In order to use a system assigned identity to authenticate with a database, the database must contain a user with the exact name of the Azure resource using the system assigned identity and be declared as an Azure AD user. The process for the creation of this user, must be performed using an Azure AD user with enough permissions to execute this action.

```
CREATE USER [managedidentitytest] FROM EXTERNAL PROVIDER;  
ALTER ROLE db_datareader ADD MEMBER managedidentitytest;  
ALTER ROLE db_datawriter ADD MEMBER managedidentitytest;
```

Listing 1. SQL queries used to create a database contained Azure AD user and grant read and write permissions.

Listing 1 shows three SQL queries, needed in this scenario to create the database user that the web application will use to connect, and then, grant that user the minimum privileges needed. The minimum permissions needed for this scenario are “read” and “write”. Those permissions are granted through the roles “db_datareader” and “db_datawriter”.

4.4.4 Configuring the storage account

To authenticate with a storage account, one Azure resource using managed identity needs to have the permissions needed for the task that will run in the storage account. In this scenario, the web application will write the activity logs in a blob container. The minimum amount of permissions that will be needed for this task, are “read” and “write” blobs. Those two permissions are grouped in the role “Storage Blob Data Contributor”.

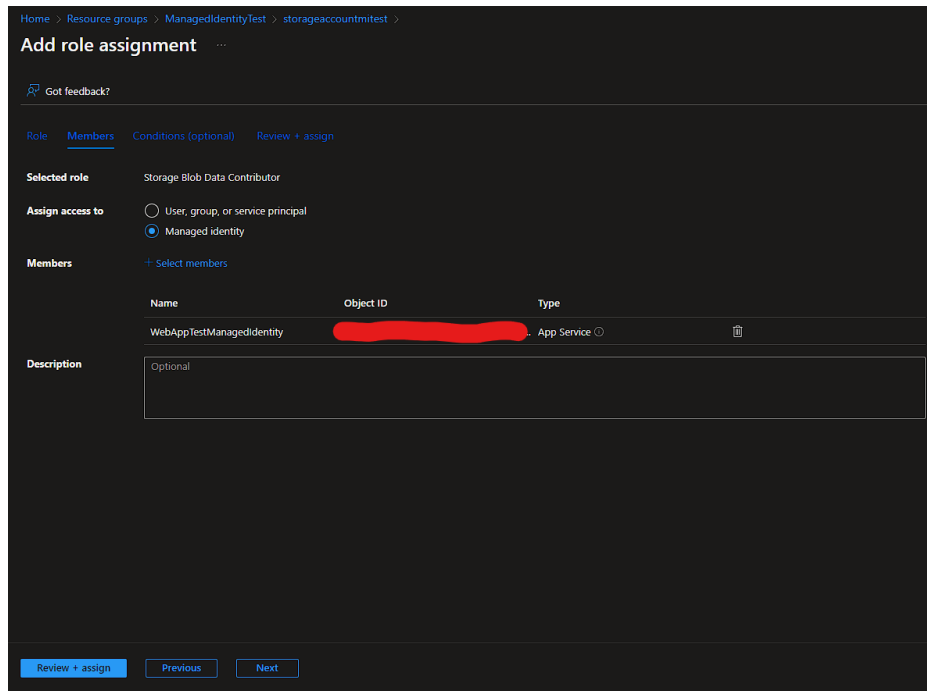


Figure 13 Access Control (IAM) Add Role Panel

Figure 13 shows the process of assigning the role, “Storage Blob Data Contributor”, to the system assigned identity, which is bounded to the web application. After finishing this process, the web application using this identity will have “read” and “write” privileges on the blob container.

4.4.5 Code implementation

Connection with the database can be established from the code using the managed identity to get an authentication token. This token replaces any password needed before in the connection string.

```

public async Task Connect()
{
    try
    {
        var tokenCredential = new DefaultAzureCredential();
        var accessToken = await tokenCredential.GetTokenAsync(
            new TokenRequestContext(scopes: new string[] {
                "https://database.windows.net/" + "/.default" }) { });

        //using Microsoft.Azure.Services.AppAuthentication is no longer
        recommended
        //string token = await (new
        AzureServiceTokenProvider()).GetAccessTokenAsync("https://database.windows.net
        /", true);

        _logger.LogWarning("TOKEN: " + accessToken.Token);
        _logger.LogInformation("Starting new connection with the database");
        connection = new SqlConnection(ConnectionString);
        connection.AccessToken = accessToken.Token;
        connection.Open();
        _logger.LogInformation("Database connection "+
        connection.State.ToString());
    }
    catch (Exception ex)
    {
        _logger.LogError(ex.ToString());
        throw ex;
    }
}

```

Listing 2. .NET Core code using system assigned managed identity to connect to a data base

Listing 2 shows an example of a code used to establish the connection with the database using the system assigned managed identity. The class “DefaultAzureCredential” is used to obtain an authentication token using the managed identity. After that, the token is used by the class “SqlConnection” which opens the connection with the database.

The connection with the storage account can be done by getting an authentication token using the managed identity. In some cases, the class used to create the connection with the storage account, is capable to handle the managed identity to open the connection.

In this scenario, the class “BlobContainerClient” has been used to connect with the blob container. This class can handle the managed identity to authenticate with the blob container.

```
string containerEndpoint =  
string.Format("https://{0}.blob.core.windows.net/{1}",  
accountName,  
containerName);  
  
// Get a credential and create a client object for the blob container.  
BlobContainerClient containerClient = new BlobContainerClient(new  
Uri(containerEndpoint), new DefaultAzureCredential());
```

Listing 3. .NET Core code using system assigned managed identity to connect to a blob container

Listing 3 shows an example code used to open a connection with blob container, using the system assigned identity of the application that runs it. The class `BlobContainerClient` is used to establish the connection with blob container. The class receives the variable “`containerEndpoint`” as parameter, which contains the Uri that refers to the storage account and the blob container, and a new instance of the class “`DefaultAzureCredential`”, which manages the system assigned identity.

4.5 User-Assigned Managed Identity Implementation

4.5.1 Overview

In this scenario, a user assigned managed identity will be created as a standalone resource. Then this identity will be associated with the web application to be used by this resource to authenticate with the database and the storage account.

A new database contained user will be created in the database. This user will have the same name that the user assigned managed identity and it will be declared as an Azure AD user. This means that the database will not store a password for this user. The user will represent an Azure AD user and Azure will handle the authentication. Once the user is created, the needed permissions have to be granted for it. In the case of the storage account, it is only necessary to grant permissions to the user assigned managed identity, which is treated as a normal Azure AD user.

4.5.2 Creating and using the user assigned identity

A user assigned identity is created independently as another Azure resource. The process is simple. It starts from the Azure portal home page, clicking the “create a resource” button and, then searching “User Assigned Managed Identity”.

The screenshot shows the 'Create User Assigned Managed Identity' panel in the Azure portal. The breadcrumb navigation at the top reads 'Home > Create a resource > User Assigned Managed Identity >'. The main heading is 'Create User Assigned Managed Identity'. Below this, there are three tabs: 'Basics' (selected), 'Tags', and 'Review + create'. The 'Project details' section includes a description: 'Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.' It contains three dropdown menus: 'Subscription *' (redacted), 'Resource group *' (ManagedIdentityTest), and 'Region *' (North Europe). Below these is the 'Instance details' section with a 'Name *' dropdown set to 'User-Assigned-Identity'. A 'Create new' link is visible under the resource group dropdown.

Figure 14 User Assigned Managed Identity Creation Panel

Figure 14 shows the creation panel for the user assigned managed identity, and the parameters used for this scenario. After choosing the subscription, it is necessary to select the resource group where the user assigned identity will be contained. Then it is possible to select the region of this resource, which should match the resource group region. Last step is to assign a unique name to the user assigned identity.

After creating a user assigned identity, it can be assigned to one or more Azure resources that supports managed identity. When the user assigned identity is associated with an Azure resource, this resource can use that identity to authenticate with other resources and use the privileges that were granted to the identity.

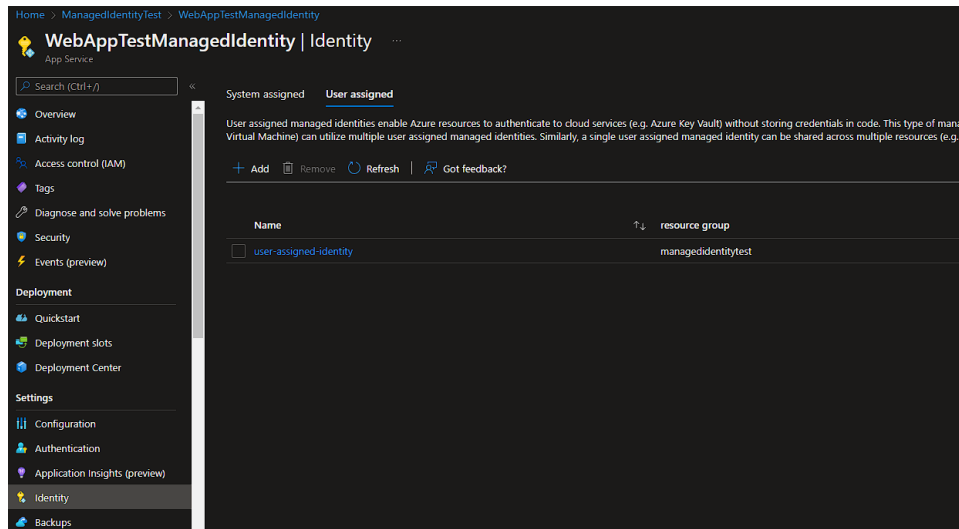


Figure 15 Panel to Add User Assigned Identities

Figure 15 shows the panel to manage the user assigned identities used by the web application. The list shows one user assigned identity associated to the web application. That identity was added from the “Add” button, which shows a panel that allows to select the user assigned identities to be added.

4.5.3 Configuring the database

In order to use a user assigned identity to authenticate with a database, the database must contain a user with the exact name of the user assigned identity resource and be declared as an Azure AD user. The process for the creation of this user, must be performed using an Azure AD user with enough permissions to execute this action.

```
CREATE USER [User-Assigned-Identity] FROM EXTERNAL PROVIDER;
ALTER ROLE db_datareader ADD MEMBER User-Assigned-Identity;
ALTER ROLE db_datawriter ADD MEMBER User-Assigned-Identity;
```

Listing 4. SQL queries used to create a database contained Azure AD user and grant read and write permissions.

Listing 4 shows three SQL queries, needed in this scenario to create the database user and grant that user the minimum privileges needed that the web

application, through the user assigned identity, will use to connect. The minimum permissions needed for this scenario are “read” and “write”. Those permissions are granted through the roles “db_datareader” and “db_datawriter”.

4.5.4 Configuring the storage account

To authenticate with a storage account, one Azure resource using managed identity needs to have only the permissions needed for the task that will run in the storage account. In this scenario, the web application will write the activity logs in a blob container. The minimum amount of permissions that will be needed for this task, are “read” and “write” blobs. Those two permissions are grouped in the role “Storage Blob Data Contributor”.

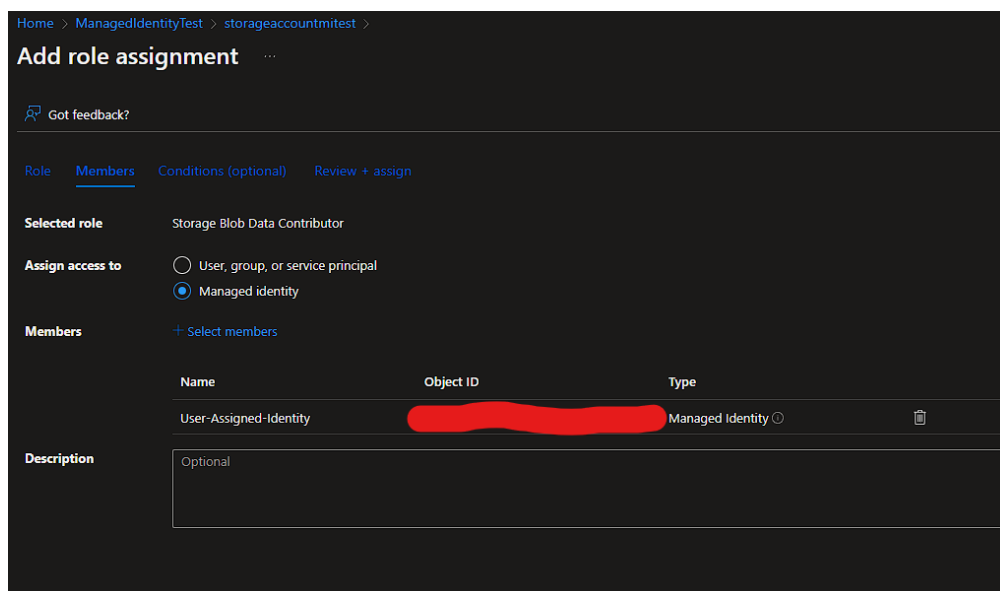


Figure 16 Access Control (IAM) Add Role Panel

Figure 16 shows the process of assigning the role, “Storage Blob Data Contributor”, to the user assigned identity, which is associated with the web application. After finishing this process, the web application using this identity will have “read” and “write” privileges on the blob container.

4.5.5 Code implementation

Developing the code, the connection with the database is created by getting an authentication token using the user assigned identity associated to the resource. When user assigned identity is used, it is necessary to specify the “Client ID” of the identity that will be used by the code.

```
public async Task Connect()
{
    try
    {
        var tokenCredential = new DefaultAzureCredential(new
DefaultAzureCredentialOptions { ManagedIdentityClientId =
Environment.GetEnvironmentVariable("managedidentitytest") });
        var accessToken = await tokenCredential.GetTokenAsync(
            new TokenRequestContext(scopes: new string[] {
                "https://database.windows.net/" + "/.default" }) { });

        //using Microsoft.Azure.Services.AppAuthentication is no longer
recomended
        //string token = await (new
AzureServiceTokenProvider()).GetAccessTokenAsync("https://database.windows.net
/", true);

        _logger.LogWarning("TOKEN: " + accessToken.Token);
        _logger.LogInformation("Starting new connection with the database");
        connection = new SqlConnection(ConnectionString);
        connection.AccessToken = accessToken.Token;
        connection.Open();
        _logger.LogInformation("Database connection "+
connection.State.ToString());
    }
    catch (Exception ex)
    {
        _logger.LogError(ex.ToString());
        throw ex;
    }
}
```

Listing 5. .NET Core code using user assigned managed identity to connect to a data base

Listing 5 shows an example code used to establish the connection with the database using the user assigned managed identity. The class “DefaultAzureCredential” is used to obtain an authentication token using the managed identity. In order to use the user assigned identity, this class needs to receive the “Client ID” of the identity as a parameter. In this example, the “Client ID” of the identity has been set as an environment variable in the Azure web application. This way, it is possible to change the user assigned identity with no

changes in the code. When the token is received, it is used by the class “SqlConnection” which opens the connection with the database.

In order to establish a connection with the blob container, in this scenario has been used the class “BlobContainerClient”. This class is capable of handling the managed identity to authenticate with the blob container.

```
string containerEndpoint =
string.Format("https://{0}.blob.core.windows.net/{1}",
accountName,
containerName);

// Get a credential and create a client object for the blob container.
BlobContainerClient containerClient = new BlobContainerClient(new
Uri(containerEndpoint), new DefaultAzureCredential(new
DefaultAzureCredentialOptions { ManagedIdentityClientId =
Environment.GetEnvironmentVariable("managedidentitytest") }));
```

Listing 6. .NET Core code using user assigned managed identity to connect to a blob container

Listing 6 shows an example code used to open a connection with blob container using the user assigned identity associated with the application that runs the code. The class BlobContainerClient is used to establish the connection with blob container. The class receives the variable “containerEndpoint” as parameter, which contains the Uri that refers to the storage account and the blob container, and a new instance of the class “DefaultAzureCredential”, which receives the “Client ID” of the identity as a parameter. The “Client ID” of the user assigned identity has been set as an environment variable in the Azure web application.

5 Conclusion

The principal objectives of this project were to explain what Azure managed identity is, and to create a simple scenario to show how to implement both types of managed identities. Those objectives have been successfully achieved using the materials and methods showed in this project.

In conclusion, Azure managed identity is a good way for companies to become password-free and be detached from the responsibility of storing and keeping passwords.

At the same time, this project has demonstrated that the implementation of Azure managed identity, in the different resources and in the code, is not complicated, and it provides additional security.

References

- 1 Becker, Riccardo. 2012. Windows Azure programming patterns for Start-ups. Packt Publishing.
- 2 Webber-Cross, Geoff. 2014. Learning Microsoft Azure. Packt Publishing.
- 3 J. Dudley, Richard & A. Duchene, Nathan. 2010. Microsoft Azure Enterprise Application Development: Enterprise Application Development. Packt Publishing.
- 4 Microsoft. What are managed identities for Azure resources? [online]. <https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/overview>
Accessed on 31/01/2022
- 5 Au, Benney. 2020. How to Use Managed Identities with Azure SQL Database [online]. <https://www.pluralsight.com/guides/how-to-use-managed-identity-with-azure-sql-database>
Accessed on 31/01/2022
- 6 Blesson, John & Issagha, BA. Azure Data Factory Security & Authentication [online]. <https://www.industry-era.com/images/pdf/Azure%20data%20Factory-Security.pdf>
Accessed on 31/01/2022
- 7 Microsoft. How managed identities for Azure resources work with Azure virtual machines [online]. <https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/how-managed-identities-work-vm>
Accessed on 05/02/2022
- 8 Microsoft. Managed identity best practice recommendations [online]. <https://docs.microsoft.com/en-us/azure/active-directory/managed-identities-azure-resources/managed-identity-best-practice-recommendations>
Accessed on 13/02/2022