# Nexus 2.0 Storage Account Configuration

## Document control

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ver | Date | Description | Updated by | Reason for Change |
| 0.1 | 2023-09-21 | Initial version | Luke Egging | Initial Doc |
|  |  |  |  |  |
|  |  |  |  |  |

## Nexus 2.0 Azure Blob Storage for the Central Database

Nexus 2.0 is all about having a single central copy of Subject data (photos plus names), which potentially thousands of independent systems can read from and use to maintain a local copy for the purposes of live operations, which in this case means real-time surveillance with a watchlist of persons of interest.

This document describes how to configure Azure Blob storage securely for production.

A screenshot of a computer

Description automatically generated

## What’s the problem?

One of the common use cases for Azure Blob Storage is to store static files that is meant to be shared externally or serve as a download site to retrieve documents. It’s an effortless way to distribute content to intended users.

It’s a common approach. It’s simple to setup, all I need to do is to create Azure Storage, set access level to public, upload files and voila I’m done. It’s just like a public FTP.

However, this method will not work in enterprise world, because it’s exposes to bad actors who could write a script to generate URLs and scrap through the storage account if he/she has learnt the file name patterns. This will then cause unwanted data leakage due to insufficient control.

At the time of writing, Azure Storage doesn’t have an out of-the-box method for securing access to Blob Storage, but there is a correct way of doing this.

This document records notes from our own implementation, using the following guides, in the following order:

1. [Secure Azure Blob Storage with Azure API Management & Managed Identities | by Marcus Tee | Marcus Tee Anytime | Medium](https://medium.com/marcus-tee-anytime/secure-azure-blob-storage-with-azure-api-management-managed-identities-b0b82b53533c)
2. [Protect API in API Management using OAuth 2.0 and Azure Active Directory - Azure API Management | Microsoft Learn](https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-protect-backend-with-aad)

This central DB looks something like this, in simplistic terms:

A diagram of a software storage container

Description automatically generated

Clients need to know four bits of information in order to successfully authenticate with the backend. Plus they need to know how to talk to the backend API.

* API endpoint (different for read/write). Setup once at implementation.
* ClientID (different for read/write). Setup once at implementation.
* Scope (different for read/write). Setup once at implementation.
* Secret (different for each unique customer system). Setup each time you onboard a new customer. Must be manually updated when it expires. Can be deleted anytime to revolk access for a particular individual. Easily managed in Azure with human-readable labels.

## Blob Storage Configuration

### Ensure Blob Storage Network access is locked down

Under: Storage account\Security + Networking\Networking

A screenshot of a computer

Description automatically generated

### Disable Blob Anonymous Access

A screenshot of a computer

Description automatically generated

## Azure API Management Configuration Steps

### Setting up Azure API Management

|  |
| --- |
| For Nexus 2.0 we actualy need to setup TWO App Gateways – one for readonly clients, another for read-write.  Action:   * You will need to repeat this section twice, once with a GET only API endpoint * And again, with a GET and PUT endpoint.   Clients that need readonly access will be given a key under the read only App Gateway, etc. |

|  |
| --- |
| This is taken word for word from [Secure Azure Blob Storage with Azure API Management & Managed Identities | by Marcus Tee | Marcus Tee Anytime | Medium](https://medium.com/marcus-tee-anytime/secure-azure-blob-storage-with-azure-api-management-managed-identities-b0b82b53533c) |

[Azure API Management](https://docs.microsoft.com/en-us/azure/api-management/api-management-key-concepts) is a combination of API Gateway and developer portal, which allows developer to access and discover APIs being published in the gateway for future usage. Administrator can then set various policies on different APIs to control access or limit the API calls.

Azure API Management is tightly integrated with Azure services such as Application Insights, Logic App, Azure App Services and Azure Functions, which simplifies API development. Of course, developers can also import existing APIs in OpenAPI, WADL or WASL format into Azure API Management.

A screenshot of a application

Description automatically generated

Interface to create or import APIs.

In our case, we will create a Blank API, which then point to our Azure Storage.

A screenshot of a computer

Description automatically generated

I called my API as AzureStorage, this is just a name, for the collections of APIs under this name.

Let’s proceed to create operations under this API. In this example, we will create a **GET**operation, which retrieve files from Azure Storage. This URL is the URL that developer will consume, so in this case, the developer will consume this API via “https://<APIM-name>.azure-api.net/”.

A screenshot of a computer

Description automatically generated

Save the operation and proceed to “Settings”. We then configure the backend web services, in this case, pointing to Azure Storage. Over here, remember to enable “subscription required” so that we can control who can access this API.

A screenshot of a computer

Description automatically generated

Note: Web service URL is the blob “endpoint” found in Storage Account \ Endpoints configuration.

We are almost there. Let’s move on to “Design” to design our API. Azure API Management allows us to configure policy, create transformation when developers consume this API, and more.

A screenshot of a computer

Description automatically generated

Design portal for complex API transformation

If you noticed, we just configured web services pointing to storage account, without the context of what container and which file. Here, we will leverage on the transformation to retrieve input, and reconstruct the URL pointing to Azure Storage. Here’s what we are going to do in our transformation:

1. Get containers and file details from request headers
2. Remove unnecessary header in API call
3. Reconstruct URL pointing to the file in Azure storage
4. Authenticate with Azure Storage via Managed Identities

Let’s add a policy at “**Inbound processing**”. There are 2 ways, either going through GUI with “**+ Add policy**”, or code-based method, clicking “**</>**”. I prefer to use code to have holistic view.

A screenshot of a computer

Description automatically generated

Let’s insert the following and remember to replace the storage account name.

This is exactly the four steps I mentioned previously. First, we define two new variables called “ContainerName” & “BlobName” and retrieve the values from request headers “Container” and “Blob” respectively. Then, we removed unwanted headers. Third step is to use C# like language to define and construct the new URL pointing to the intended container and file. Lastly, API Management should authenticate with Azure Storage with Managed Identities instead of SAS token.

Save the policy and we are done creating an API operation to retrieve files from our storage account.

### Setting up Managed Identities and Authentication for Azure Storage

|  |
| --- |
| Following the same section in [Secure Azure Blob Storage with Azure API Management & Managed Identities | by Marcus Tee | Marcus Tee Anytime | Medium](https://medium.com/marcus-tee-anytime/secure-azure-blob-storage-with-azure-api-management-managed-identities-b0b82b53533c) |

Managed Identities is a powerful option to secure our cloud native services because it removed the needs of maintaining service account, and password. Often, a lot of cybersecurity incident is due to stolen passwords. With Managed Identities, we removed the needs of maintaining a service account, yet able to secure and authenticate the request. Here’s how it works:

A diagram of a cloud

Description automatically generated

As you can see, we don’t need to configure additional service account, as Azure API Management is registered as an object in Azure AD, and it will authenticate itself to retrieve authentication token, and use this token to authenticate with Azure Storage. Azure Storage will then recognize it’s an authenticated request, hence completing this request.

To do this, we need to perform two steps, firstly, enable Managed Identities in Azure API Management, and secondly, configure Azure Storage to use Azure AD authentication.

Managed Identities registration is straightforward, simple toggle the selection in Azure API Management.

A screenshot of a computer

Description automatically generated

The next step is non-trivial. There are two type of access control for Azure storage, which is at resource level (for administrative task such as modify storage account permission etc), and role-based access control at file level, and in this case, we are configuring the latter aspect.

To configure this, navigate to the respective container within your storage account. You will see “Access Control” which is different from resource level RBAC.

A screenshot of a computer

Description automatically generated

In terms of roles, you will find roles specific to storage account data, such as “Storage Blob Data Owner”, “Storage Blob Data Reader” etc, and you can see the full list [here](https://docs.microsoft.com/en-us/azure/role-based-access-control/built-in-roles). In my case, I assign “Storage Blob Data Owner” to my Azure API Management, so that it can has full access. Depending on the policies, feel free to select the minimal control to fulfill least of privilege principal.

A screenshot of a computer

Description automatically generated

Now, the last step, which is configure Azure Storage to use Azure AD to authenticate, instead of Access token. To do that, navigate back to “Overview” again, and you will see **Authentication method**. Simply switch to Azure AD authentication.

A screenshot of a computer

Description automatically generated

After switching it, you may receive a warning banner saying, “You do not have permissions to list the data….”, that’s is okay and don’t worry on the message.

A screenshot of a computer

Description automatically generated

That’s all we need to do! Now, let’s head back to our Azure API Management to test this.

## Additional Security factor – API Subscription Key

We can add a layer of security to clients by giving them the API Key that is locked to a particular API, e.g. the read-only API Key will ONLY work with the read API.

This is configured in the API Management (API Gatway) in Azure as shown here:

A screenshot of a computer

Description automatically generated

The “Primary key” must then be given to clients to help them authenticate, as an additional factor to the OAuth 2.0 method.

As you can see below, this needs to be configured (ticked) for each API (readonly and ReadWrite).

A screenshot of a computer

Description automatically generated

## Oops – forgot to update the API Gateway (2nd Article)

This article [Secure Azure Blob Storage with Azure API Management & Managed Identities | by Marcus Tee | Marcus Tee Anytime | Medium](https://medium.com/marcus-tee-anytime/secure-azure-blob-storage-with-azure-api-management-managed-identities-b0b82b53533c) forgot that while the test in Azure will work, if you hit the endpoint with Postman without the SAS token, it will fail:

A screenshot of a computer

Description automatically generatedTo fix this, we need to follow the second article:

[Protect API in API Management using OAuth 2.0 and Azure Active Directory - Azure API Management | Microsoft Learn](https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-protect-backend-with-aad)

## Register an application in Azure AD to represent the API

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| As with API Gateways , For Nexus 2.0 we actualy need to setup TWO App Registrations – one for readonly clients, another for read-write.  Action:   * You will need to repeat this section twice, once with the GET only API Gatway * And again, with the GET/PUT API GateWay   Clients that need readonly access will be given a key under the read only app registration, etc.   |  | | --- | | New – ReadOnly Config Endpoint: https://necnexus2apitest.necswscloud.com/AAD/  ClientID: 1478c6b0-bc5b-4c5a-acae-afff3db9abc2 (from app registration)  ApplicaitonID Uri: api://1478c6b0-bc5b-4c5a-acae-afff3db9abc2 (from app registration Expose an API)  Client Secret: NAP8Q~q3ULvat.lWnPJ2VqgbWhplM~-9grR-KddM. (from app registration, needs to be maintained)  ReadOnly Scope: api://1478c6b0-bc5b-4c5a-acae-afff3db9abc2/ReadOnly (from app registration)  Auth grant flow: <https://necnexus2apimanagement-apim.developer.azure-api.net/signin-oauth/code/callback/readonlyauth> (generated from creation of oAuth server in API Management service) | | New – ReadWrite Config Endpoint: https://necnexus2apitest.necswscloud.com/AAD-READWRITE/  ClientID: b9a59f71d1174da59ca5f7bf868a6c4b (from app registration)  ApplicaitonID Uri: api://e08fde3b-4399-48c3-84b3-c2a7dd55b21c (from app registration Expose an API)  Client Secret: JQG8Q~c9.WgMo1kiwTxtfIpnbCPKngc3cu7xhbj. (from app registration, needs to be maintained)  ReadWrite Scope: api://e08fde3b-4399-48c3-84b3-c2a7dd55b21c/ReadWrite (from app registration) Auth grant flow: <https://necnexus2apimanagement-apim.developer.azure-api.net/signin-oauth/code/callback/readonlyauth> (generated from creation of oAuth server in API Management service) |  Luke-ApiGateWay-ReadOnlyClients  |  |  | | --- | --- | | ClientID | 6140ae32-fae1-4e71-991f-059e701e5e5c | | tenantID | 71579be7-5e6f-4786-82a9-63d2090a1f06 | | Redirect | https://apimanagementnecswstest.developer.azure-api.net/signin-oauth/code/callback/lukeapitestauthserver | | Client Secret 1  APIM | NAP8Q~q3ULvat.lWnPJ2VqgbWhplM~-9grR-KddM |   Luke-ApiGateWay-ReadWriteClients   |  |  | | --- | --- | | ClientID | 6140ae32-fae1-4e71-991f-059e701e5e5c | | tenantID | 71579be7-5e6f-4786-82a9-63d2090a1f06 | | Redirect | https://apimanagementnecswstest.developer.azure-api.net/signin-oauth/code/callback/lukeapitestauthreadonly | | Client Secret 1  ApiGateWay  (APIM) | JQG8Q~c9.WgMo1kiwTxtfIpnbCPKngc3cu7xhbj. | |

Using the Azure portal, protect an API with Azure AD by first registering an application that represents the API.

For details about app registration, see [Quickstart: Configure an application to expose a web API](https://learn.microsoft.com/en-us/azure/active-directory/develop/quickstart-configure-app-expose-web-apis).

1. In the [Azure portal](https://portal.azure.com/), search for and select **App registrations**.
2. Select **New registration**.
3. When the **Register an application page** appears, enter your application's registration information:
   * In the **Name** section, enter a meaningful application name that will be displayed to users of the app, such as backend-app.
   * In the **Supported account types** section, select an option that suits your scenario. (this has to be set to: “**Accounts in any organizational directory (Any Microsoft Entra ID tenant - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)”**
4. Leave the [**Redirect URI**](https://learn.microsoft.com/en-us/azure/active-directory/develop/reply-url) section empty. (we will fill this later)
5. Select **Register** to create the application.
6. On the app **Overview** page, find the **Application (client) ID** value and record it for later.
7. Under the **Manage** section of the side menu, select **Expose an API** and set the **Application ID URI** with the default value. If you're developing a separate client app to obtain OAuth 2.0 tokens for access to the backend-app, record this value for later.
8. Select the **Add a scope** button to display the **Add a scope** page:
   * Enter a new **Scope name**, **Admin consent display name**, and **Admin consent description**.
   * Make sure the **Enabled** scope state is selected.
9. Select the **Add scope** button to create the scope.
10. Repeat the previous two steps to add all scopes supported by your API.
11. Once the scopes are created, make a note of them for use later.

Also create a new “Client secret” under certificates and secrets. We will need this later

A screenshot of a computer

Description automatically generated

ClientID d43ddf9e-f614-4ebe-a0cf-0761e6c79e1f

Client secret: .NL8Q~DRZb13kUti0dl-IxrtR.0XvHTiVbGXjcTw

ApplicaitonID Uri: api://d43ddf9e-f614-4ebe-a0cf-0761e6c79e1f

Readonly Scope: api://d43ddf9e-f614-4ebe-a0cf-0761e6c79e1f/ReadOnly

tenantID 71579be7-5e6f-4786-82a9-63d2090a1f06

### Configure an OAuth 2.0 authorization server in API Management

Now got to this article [Authorize test console of API Management developer portal using OAuth 2.0 - Azure API Management | Microsoft Learn](https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-oauth2) section ‘Configure an OAuth 2.0 authorization server in API Management’

Client registration page url: https://login.live.com/

Authorisation endpoint url: <https://login.microsoftonline.com/71579be7-5e6f-4786-82a9-63d2090a1f06/oauth2/v2.0/authorize> (GUID IS your TENANT ID)

Token endpoint url: <https://login.microsoftonline.com/71579be7-5e6f-4786-82a9-63d2090a1f06/oauth2/v2.0/token>

Client id: d43ddf9e-f614-4ebe-a0cf-0761e6c79e1f

Client secret: (take from appropriate app registration for APIM, e.g. Luke-ApiGateWay-ReadWriteClients or Luke-ApiGateWay-ReadOnlyClients, above)

Redirect URI – auth code grant flow: https://apimanagementnecswstest.developer.azure-api.net/signin-oauth/code/callback/lukeapitestauthserver

Redirect URI implicit grant flow: <https://apimanagementnecswstest.developer.azure-api.net/signin-oauth/implicit/callback>

1. In the [Azure portal](https://portal.azure.com/), navigate to your API Management instance.
2. Under **Developer portal** in the side menu, select **OAuth 2.0 + OpenID Connect**.
3. Under the **OAuth 2.0** tab, select **+ Add**.
4. Enter a name and an optional description in the **Name** and **Description** fields.

**Note**

These fields identify the OAuth 2.0 authorization server within the current API Management service. Their values do not come from the OAuth 2.0 server.

1. Enter the **Client registration page URL** - for example, https://contoso.com/login. This page is where users can create and manage their accounts, if your OAuth 2.0 provider supports user management of accounts. The page varies depending on the OAuth 2.0 provider used.

If your OAuth 2.0 provider doesn't have user management of accounts configured, enter a placeholder URL here such as the URL of your company, or a URL such as http://localhost.

1. The next section of the form contains the **Authorization grant types**, **Authorization endpoint URL**, and **Authorization request method** settings.
   * Select one or more desired **Authorization grant types**. For this example, select **Authorization code** (the default). [Learn more](https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-oauth2#authorization-grant-types)
   * Enter the **Authorization endpoint URL**. You can obtain the endpoint URL from the **Endpoints** page of one of your app registrations. For a single-tenant app in Azure AD, this URL will be similar to one of the following URLs, where {aad-tenant} is replaced with the ID of your Azure AD tenant.

Using the v2 endpoint is recommended; however, API Management supports both v1 and v2 endpoints.

https://login.microsoftonline.com/{aad-tenant}/oauth2/v2.0/authorize (v2)

https://login.microsoftonline.com/{aad-tenant}/oauth2/authorize (v1)

* + The **Authorization request method** specifies how the authorization request is sent to the OAuth 2.0 server. Select **POST**.

1. Specify **Token endpoint URL**, **Client authentication methods**, **Access token sending method**, and **Default scope**.
   * Enter the **Token endpoint URL**. For a single tenant app in Azure AD, it will be similar to one of the following URLs, where {aad-tenant} is replaced with the ID of your Azure AD tenant. Use the same endpoint version (v2 or v1) that you chose previously.

https://login.microsoftonline.com/{aad-tenant}/oauth2/v2.0/token (v2)

https://login.microsoftonline.com/{aad-tenant}/oauth2/token (v1)

* + If you use **v1** endpoints, add a body parameter:  
    \* Name: **resource**.  
    \* Value: the back-end app **Application (client) ID**.
  + If you use **v2** endpoints:  
    \* Enter the back-end app scope you created in the **Default scope** field.  
    \* Set the value for the [accessTokenAcceptedVersion](https://learn.microsoft.com/en-us/azure/active-directory/develop/reference-app-manifest" \l "accesstokenacceptedversion-attribute) property to 2 in the [application manifest](https://learn.microsoft.com/en-us/azure/active-directory/develop/reference-app-manifest) for both the backend-app and the client-app registrations.
  + Accept the default settings for **Client authentication methods** and **Access token sending method**.

1. In **Client credentials**, enter the **Client ID** and **Client secret**, which you obtained during the creation and configuration process of your client-app.
2. After the **Client ID** and **Client secret** are specified, the **Redirect URI** for the **authorization code** is generated. This URI is used to configure the redirect URI in your OAuth 2.0 server configuration.

In the developer portal, the URI suffix is of the form:

* + /signin-oauth/code/callback/{authServerName} for authorization code grant flow
  + /signin-oauth/implicit/callback for implicit grant flow

Copy the appropriate Redirect URI to the **Authentication** page of your client-app registration. In the app registration, select **Authentication** > **+ Add a platform** > **Web**, and then enter the Redirect URI.

1. If **Authorization grant types** is set to **Resource owner password**, the **Resource owner password credentials** section is used to specify those credentials; otherwise you can leave it blank.
2. Select **Create** to save the API Management OAuth 2.0 authorization server configuration.
3. [Republish](https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-developer-portal-customize#publish) the developer portal. (click on portal overview and then click “publish” button

### Configure an API to use OAuth 2.0 user authorization

After saving the OAuth 2.0 server configuration, configure an API or APIs to use this configuration.

**Important**

* Configuring OAuth 2.0 user authorization settings for an API enables API Management to acquire a token from the authorization server when you use the test console in the Azure portal or developer portal. The authorization server settings are also added to the API definition and documentation.
* For OAuth 2.0 authorization at runtime, the client app must acquire and present the token and you need to configure token validation in API Management or the backend API. For an example, see [**Protect an API in Azure API Management using OAuth 2.0 authorization with Azure Active Directory**](https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-protect-backend-with-aad).

1. Select **APIs** from the **API Management** menu on the left.
2. Select the name of the desired API and select the **Settings** tab. Scroll to the **Security** section, and then select **OAuth 2.0**.
3. Select the desired **Authorization server** from the drop-down list, and select **Save**.

A screenshot of a computer

Description automatically generated

And don’t forget to turn off Subscription required on this screen!

A screenshot of a computer

Description automatically generated

## Developer portal - test the OAuth 2.0 user authorization

Let’s test with Postman. To do this you have to configure a redirect in the app registration to allow post man to authenticate:

A screenshot of a computer

Description automatically generated

Follow this guide: [Call an ASP.NET Core web API with Postman - Microsoft Entra | Microsoft Learn](https://learn.microsoft.com/en-us/azure/active-directory/develop/howto-call-a-web-api-with-postman?tabs=dotnet6&pivots=no-api)

abced

# Config to allow Blob Listing

You will get this error by default.

A screenshot of a computer

Description automatically generated

By changing container access to anonymous for read containers and blobs, then you can list

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

As you can see here:

A screenshot of a computer

Description automatically generated

## How to do this in APIM and Azure AD

My problem was that in my READWRITE API, “GET /” was already taken, and Azure Blob doesn’t work with APIM with a customer url pattern (e.g. “GET /list-blobs”)

I had to add the url-rewrite back to “GET /” to make this work:

<policies>

    <inbound>

        <rewrite-uri template="/" />

        <set-variable name="ContainerName" value="@(context.Request.Headers.GetValueOrDefault("Container"))" />

        <base />

        <authentication-managed-identity resource="https://storage.azure.com/" />

        <set-header name="Container" exists-action="delete" />

        <set-header name="Ocp-Apim-Subscription-Key" exists-action="delete" />

        <set-header name="Sec-Fetch-Site" exists-action="delete" />

        <set-header name="Sec-Fetch-Mode" exists-action="delete" />

        <set-header name="Sec-Fetch-Dest" exists-action="delete" />

        <set-header name="Accept" exists-action="delete" />

        <set-header name="Accept-Encoding" exists-action="delete" />

        <set-header name="Referer" exists-action="delete" />

        <set-header name="X-Forwarded-For" exists-action="delete" />

        <set-header name="x-ms-version" exists-action="override">

            <value>@{string version = "2019-12-12"; return version;}</value>

        </set-header>

        <set-backend-service base-url="@{

string containerName = context.Variables.GetValueOrDefault<string>("ContainerName");

return String.Format("https://stnecnexus2.blob.core.windows.net/{0}", containerName);

}" />

        <set-query-parameter name="restype" exists-action="override">

            <value>container</value>

        </set-query-parameter>

        <set-query-parameter name="comp" exists-action="override">

            <value>list</value>

        </set-query-parameter>

    </inbound>

    <backend>

        <base />

    </backend>

    <outbound>

        <base />

    </outbound>

    <on-error>

        <base />

    </on-error>

</policies>

# Auditing User Activities

User Activities (via the API) are audited configuring Application Insights on the API-M (API Gateway). This is further described in [Nexus Open Watchlist Sync v0.3.docx](https://necsws.sharepoint.com/:w:/r/sites/msteams_d74674/Shared%20Documents/General/Business%20Cases/Nexus%20Open%20Watchlist%20Sync%20v0.3.docx?d=wcd3d16347fe54f1d92ac1c99458e213c&csf=1&web=1&e=zI1bZh) (under Biometrics \ business cases in 2023-12)