# The Tenant Management Application

The [TenantManagement](https://github.com/PowerBiDevCamp/TenantManagement/tree/main/TenantManagement) application is a sample .NET 5 application which demonstrates how to manage service principals within a large-scale Power BI embedding environment with 1000's of customer tenants. Let's start by explaining what is meant by a tenant.

If you have worked with Azure AD, the word **"tenant"** might make you think of an Azure AD tenant. However, the concept of a tenant is different for this sample application. In this context, each tenant represents a customer for which you are embedding Power BI reports using the app-owns-data embedding model. In order to manage a multi-tenant environment, you must create a separate tenant for each customer. Provisioning a new customer tenant for Power BI embedding typically involves writing code to create a Power BI workspace, import a PBIX file, patch datasource credentials and start a dataset refresh operation.

The problem that **TenantManagement** application addresses is a Power BI Service limitation which restricts users and service principals from being a member of more than 1000 workspaces. If you are implementing app-owns-data embedding in an application which uses a single service principal, Microsoft will only support you in creating up to 1000 workspaces.

The **TenantManagement** application demonstrates how to work around the 1000 workspace limitation by implementing a service principal pooling scheme. Here is how it works. Each service principal can support up to 1000 workspaces. Therefore, creating a service principal pool of 10 service principals makes it possible to create and manage 10,000 customer tenant workspaces in a fashion that is supported by Microsoft.

In addition to implementing a service principal pooling scheme, the **TenantManagement** application also demonstrates how to create and manage a separate service principal for each customer tenant workspace. An application design which maintains a one-to-one relationship between service principals and customer tenant workspaces is what Microsoft recommends as a best practice because it provides the greatest amount of isolation especially with respect datasource credentials.

You can follow the steps in this document to set up the **TenantManagement** application for testing. To complete these steps, you will require a Microsoft 365 tenant in which you have permissions to create and manage Azure AD applications and security groups. You will also need Power BI Service administrator permissions to configure Power BI settings to give service principals to ability to access the Power BI Service API. If you do not have a Microsoft 365 environment for testing, you can create one for free by following the steps in [Create a Development Environment for Power BI Embedding](https://github.com/PowerBiDevCamp/Camp-Sessions/raw/master/Create%20Power%20BI%20Development%20Environment.pdf).

## Setting up your development environment

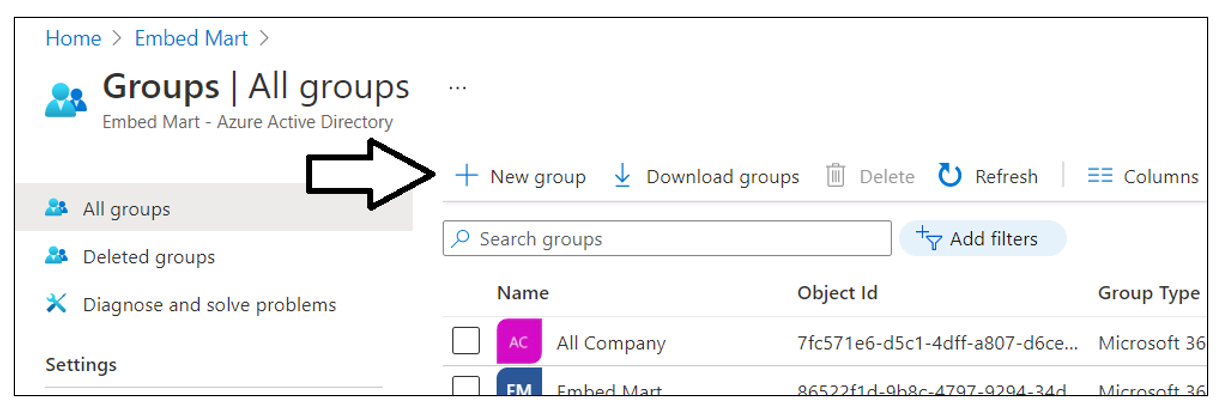
To set up the TenantManagement application doe testing, you will need to configure a Microsoft 365 envviroment with the following tasks.

1. Create a Security Group in Azure AD named Power BI Apps
2. Configure Power BI Tenant-Level Settings for Service Principal Access
3. Create the Azure AD Application for the TenantManagement Application

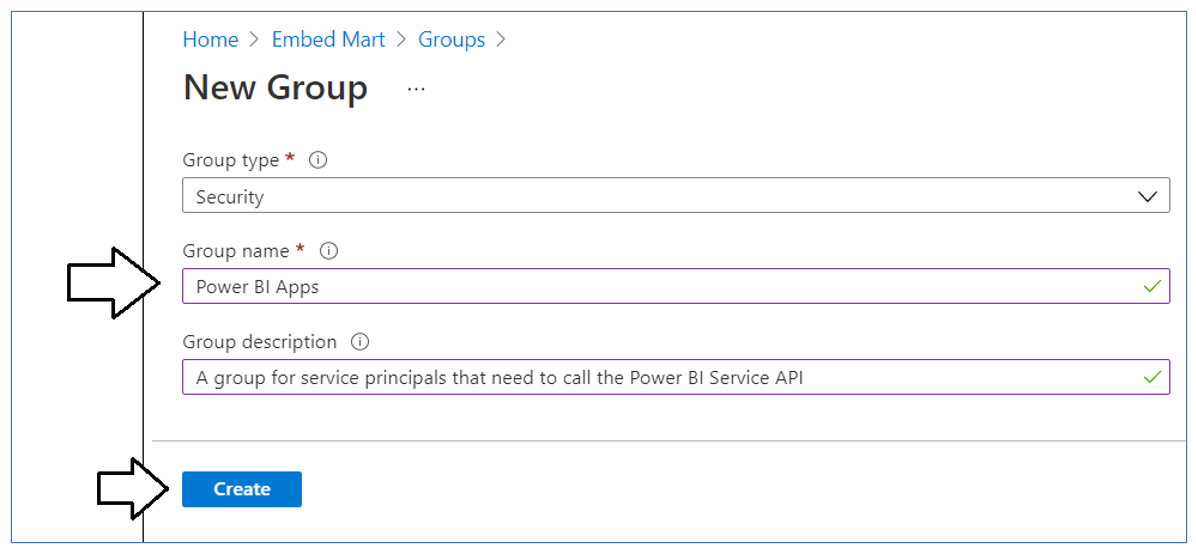
The following three sections will step through each of these setup tasks.

### Create an Azure AD security group named Power BI Apps

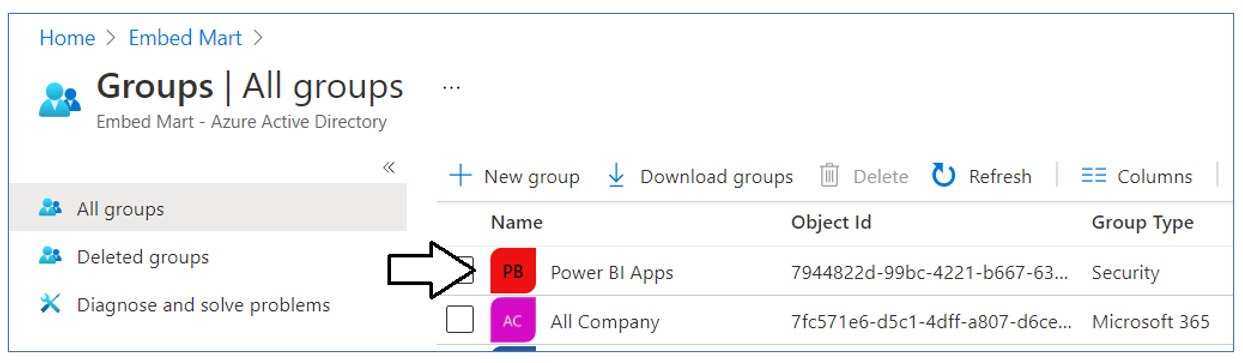
Begin by navigating to the [Groups management page](https://portal.azure.com/#blade/Microsoft_AAD_IAM/GroupsManagementMenuBlade/AllGroups) in the Azure portal. Once you get to the **Groups** page in the Azure portal, click the **New group** link.



In the **New Group** dialog, Select a **Group type** of **Security** and enter a **Group name** of **Power BI Apps**. Click the **Create** button to create the new Azure AD security group



Verify that you can see the new security group named **Power BI Apps** on the Azure portal **Groups** page.



### Configure Power BI tenant-level settings for service principal access

Next, you need you enable a tenant-level setting for Power BI named **Allow service principals to use Power BI APIs**. Navigate to the Power BI Service admin portal at <https://app.powerbi.com/admin-portal>. In the Power BI Admin portal, click the **Tenant settings** link on the left.

Graphical user interface, application

Description automatically generated

Move down in the **Developer settings** section and expand the **Allow service principals to use Power BI APIs** section.

Graphical user interface, application

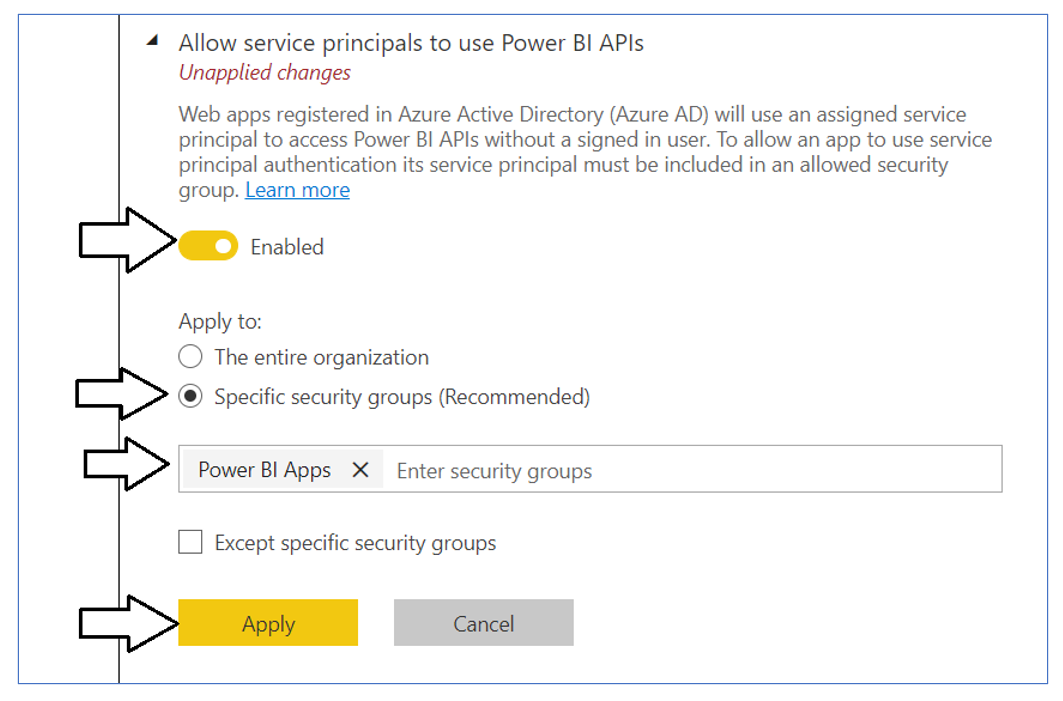
Description automatically generated

Note that the **Allow service principals to use Power BI APIs** setting is initially set to **Disabled**.

Graphical user interface, text, application, email

Description automatically generated

Change the setting to **Enabled**. After that, set the **Apply to** setting to **Specific security groups** and add the **Power BI Apps** security group as shown in the screenshot below. Click the **Apply** button to save your configuration changes.

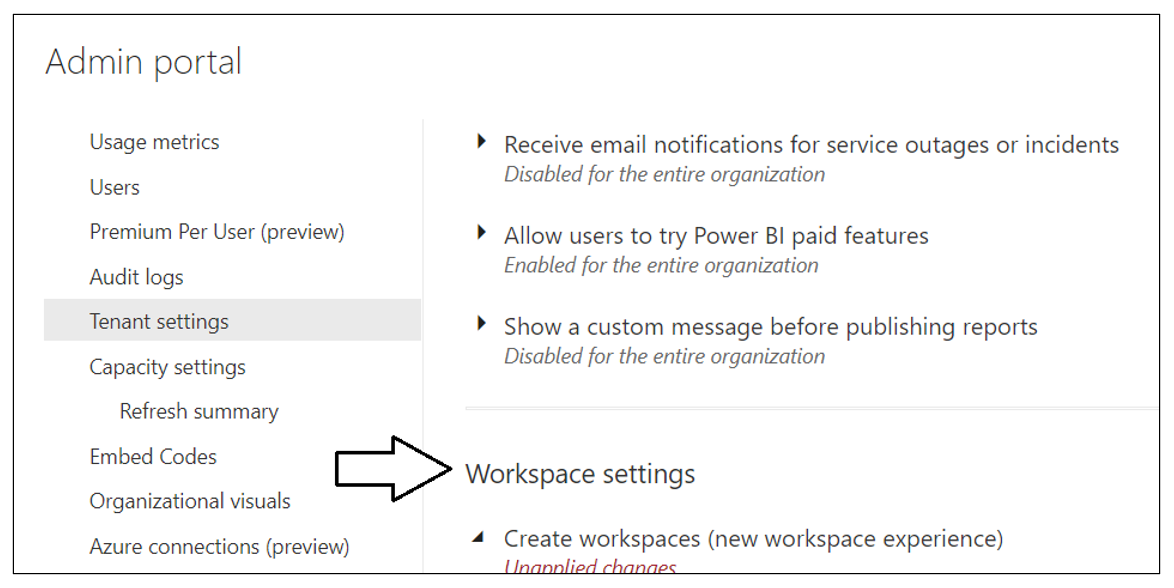


You will see a notification indicating it might take up to 15 minutes to apply these changes to the organization.

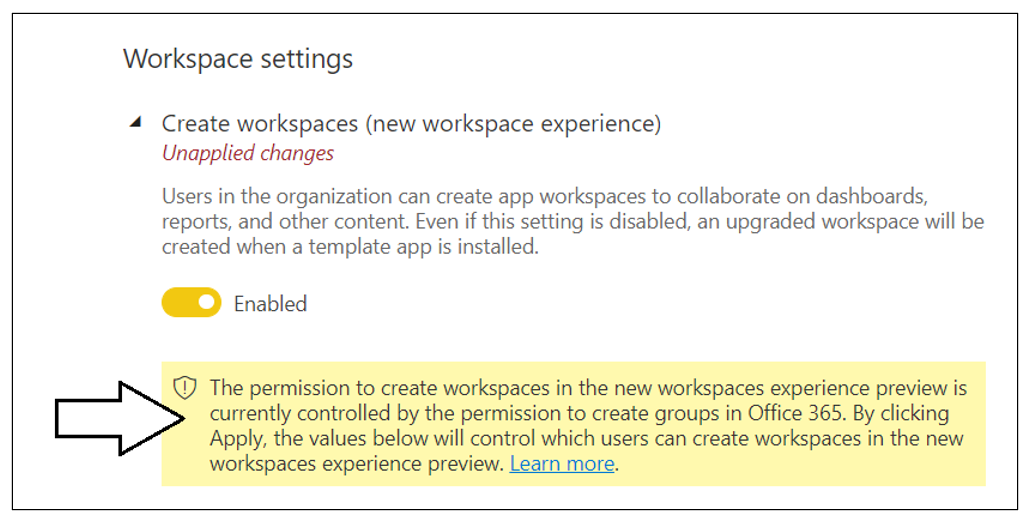
Text

Description automatically generated with medium confidence

Now scroll upward in the **Tenant setting** section of the Power BI admin portal and locate the **Workspace settings** section.



Note that a new Power BI tenant has an older policy where only users who have the permissions to create Office 365 groups can create new Power BI workspaces. You must reconfigure this setting so that service principals in the **Power BI Apps** group will be able to create new workspaces.



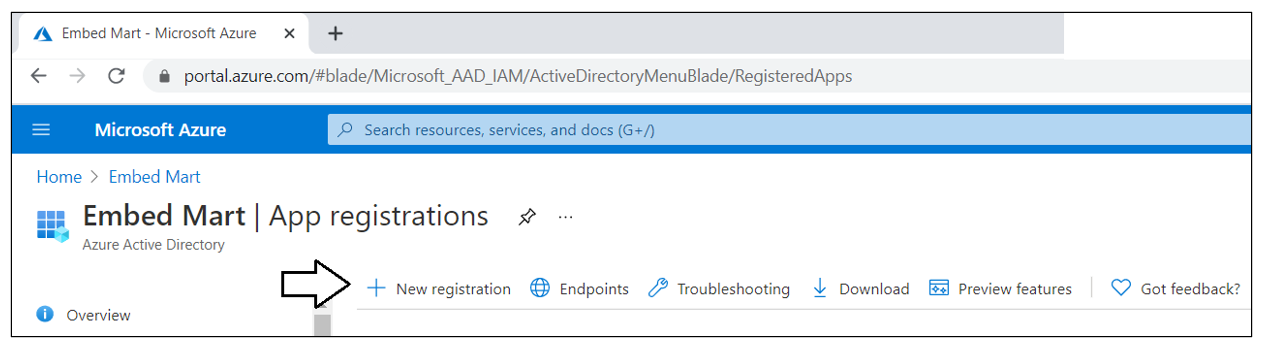
In the **Workspace settings** section, set the **Apply to** setting to **The entire organization**. Click the **Apply** button to save your configuration changes.



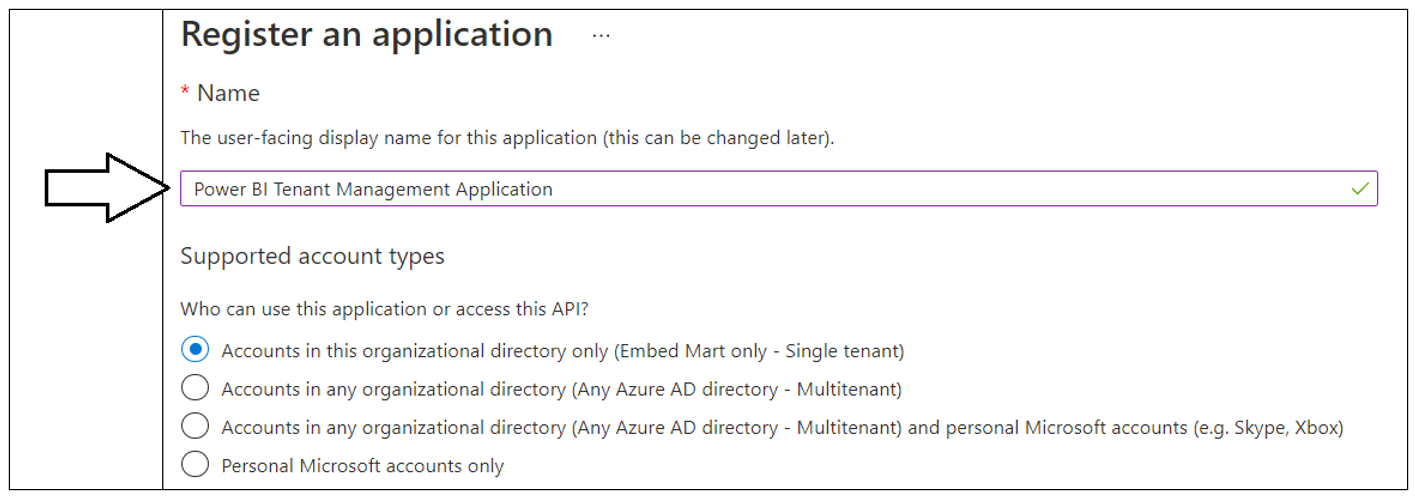
You have now completed the configuration of Power BI tenant-level settings.

### Create the Azure AD Application for the TenantManagement Application

When you login to the Azure portal to create the new Azure AD application, make sure you log in using a user account in the same tenant which contains the Power BI reports you'd like to embed. Begin by navigating to the [App registration](https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps) page in the Azure portal and click the **New registration** link.



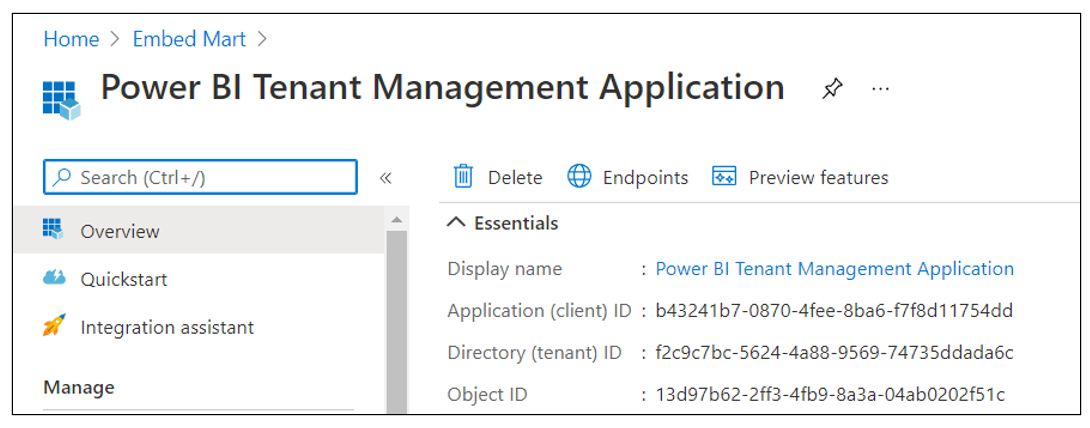
On the **Register an application** page, enter an application name such as **Power BI Tenant Management Application** and accept the default selection for **Supported account types** of **Accounts in this organizational directory only**.



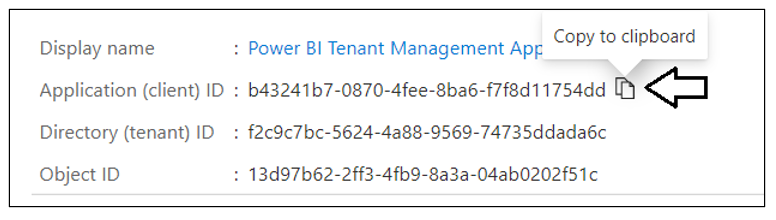
In the **Redirect URI** section leave the default selection of **Web** in the dropdown box. In the textbox to the right of the dropdown, enter a Redirect URI of **https://localhost:44300/signin-oidc**. Click the **Register** button to create the new Azure AD application.



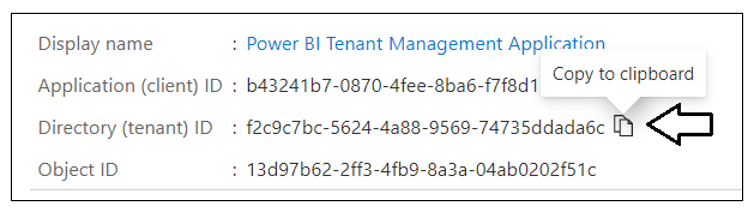
After creating a new Azure AD application in the Azure portal, you should see the Azure AD application overview page which displays the **Application ID**. Note that the ***Application ID*** is often called the ***Client ID***, so don't let this confuse you. You will need to copy this Application ID and store it so you can use it later to configure the project's support for Client Credentials Flow.



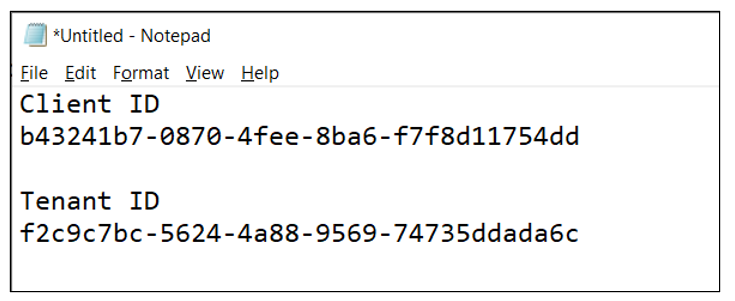
Copy the **Client ID** (aka Application ID) and paste it into a text document so you can use it later in the setup process. Note that this is the **Client ID** value that will be used by **TenantManagement** project to authenticate users.



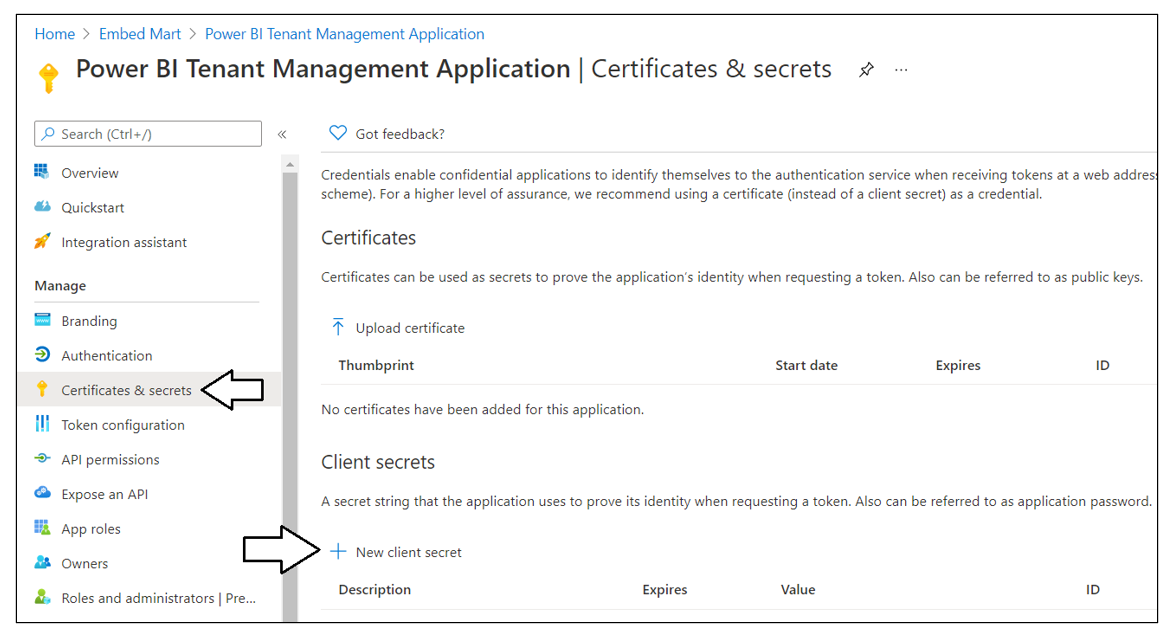
Next, repeat the same step by copying the **Tenant ID** and copying that into the text document as well.



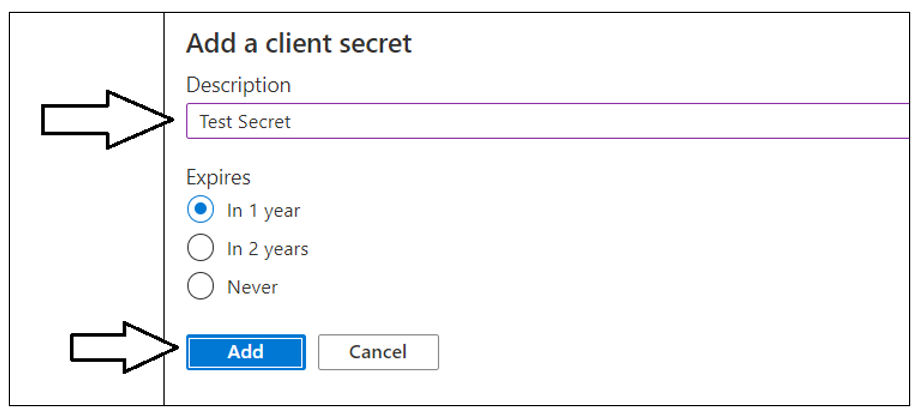
Your text document should now contain the **Client ID** and **Tenant ID** as shown in the following screenshot.



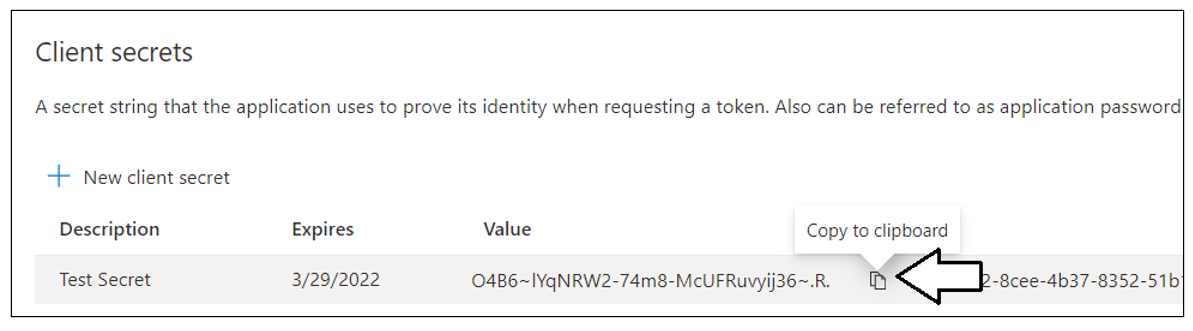
Next, you need to create a Client Secret for the application. Click on the **Certificates & secrets** link in the left navigation to move to the **Certificates & secrets** page. On the **Certificates & secrets** page, click the **New client secret** button as shown in the following screenshot.



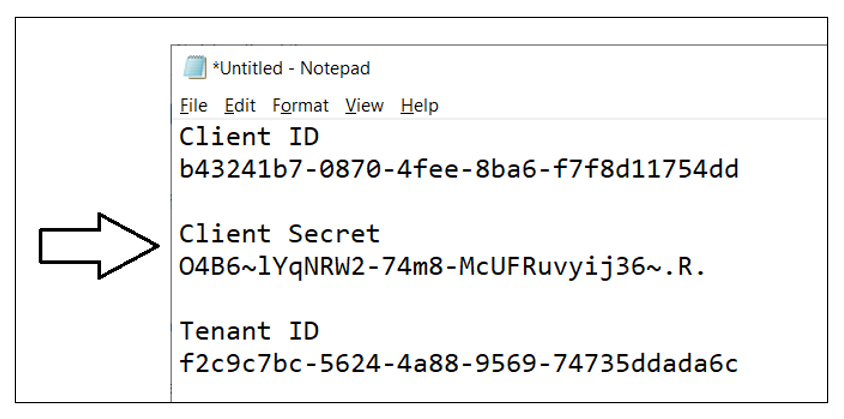
In the **Add a client secret** dialog, add a text description such as **Test Secret** and then click the **Add** button to create the new Client Secret.



Once you have created the Client Secret, you should be able to see its **Value** in the **Client secrets** section. Click on the **Copy to clipboard** button to copy the Client Secret into the clipboard.



Paste the **Client Secret** into the same text document with the **Client ID** and **Tenant ID**.



## Testing the Tenant Management project with Visual Studio 2019

In order to run and test the **TenantManagement** project on a developer workstation, you must install the .NET 5 SDK and Visual Studio 2019. While this document will walk through the steps of opening and running the **TenantManagement** project using Visual Studio 2019, you can also open and run the project using Visual Studio Code if you prefer that IDE. Here are links to download this software if you need them.

1. .NET 5 SDK – [[download](https://dotnet.microsoft.com/download/dotnet/5.0)]
2. Visual Studio 2019 – [[download](https://visualstudio.microsoft.com/downloads/)]
3. Visual Studio Code – [[download](https://code.visualstudio.com/Download)]

### Download the Source Code

The source code for the **TenantManagement** project is maintained in a GitHib repository at the following URL.

* <https://github.com/PowerBiDevCamp/TenantManagement>

You can download the **TenantManagement** project source files in a single ZIP archive using [this link](https://github.com/PowerBiDevCamp/TenantManagement/archive/refs/heads/main.zip). If you are familiar with the **git** utility, you can clone the project source files to your local developer workstation using the following **git** command.

git clone <https://github.com/PowerBiDevCamp/TenantManagement.git>

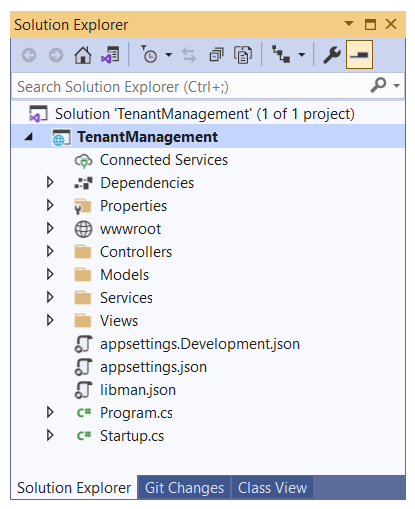
Once you have downloaded the source files for the **TenantManagement** repository to your developer workstation, you will see there is a top-level project folder named **TenantManagement** which contains several files including a solution file named **TenantManagement.sln** and a project file named **TenantManagement.csproj**.

Graphical user interface, application

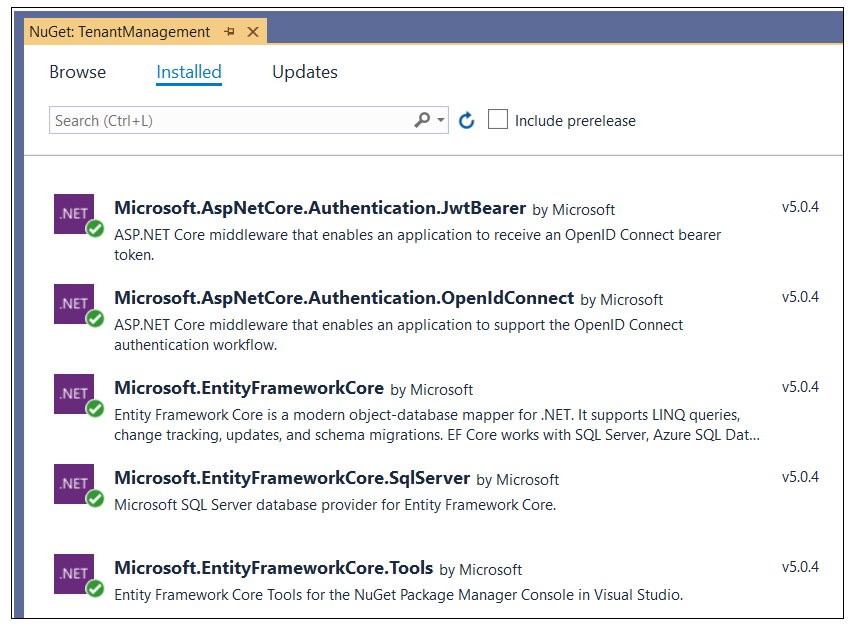
Description automatically generated

### Open the Project in Visual Studio 2019

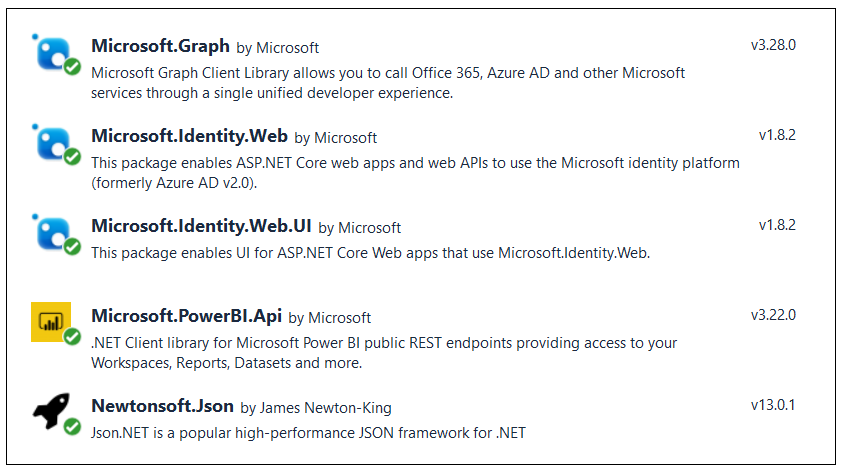
Launch Visual Studio 2019 and use the **File > Open > Project/Solution** menu command to open the solution file named **TenantManagement.sln**. This development project has been built as a .NET 5 MVC Web Application as shown in the following screenshot.



Let's quickly review the NuGet packages that have been installed in the **TenantManagement** project. There are several NuGet packages which add Entity Framework support which make it possible to quickly create the SQL Server database associated with this project.



There are several packages included to add Azure AD authentication support including **Microsoft.Identity.Web** and **Microsoft.Identity.Web.UI**. The package named **Microsoft.Graph** has been included to support .NET programming with the Microsoft Graph API. The package named **Microsoft.PowerBI.Api** has been included to support .NET programming against the Power BI REST API.

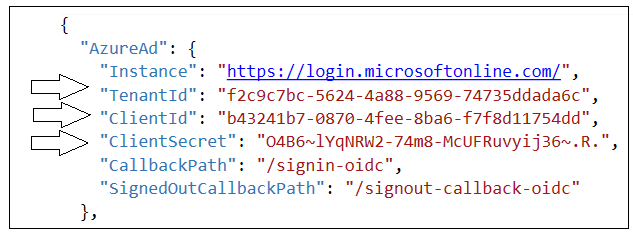


### Update application settings in the appsettings.json file

Before you can run the application in the Visual Studio debugger, you must update several critical application settings in the **appsettings.json** file. Open the **appsettings.json** file and examine the JSON content inside. There is three important sections named **AzureAd**, **TenantManagementDB** and **DemoSettings**.



Inside the **AzureAd** section, update the **TenantId**, **ClientId** and **ClientSecret** with the data you collected when creating the Azure AD application named **Power BI Tenant Management Application.**



If you are using Visual Studio 2019, you can leave the database connection string the way it is with the **Server** setting of **(localdb)\\MSSQLLocalDB**. You can change this connection string to point to a different server if you'd rather create the project database named **TenantManagementDB** in a different location.



In the **DemoSettings** section there is a property named **AdminUser**. The reason that this property exists has to with you being able to see Power BI workspaces as they are created by service principals. Update the **AdminUser** property setting with your Azure AD account name so that you will be added as an Admin member to any Power BI workspaces created by this application.

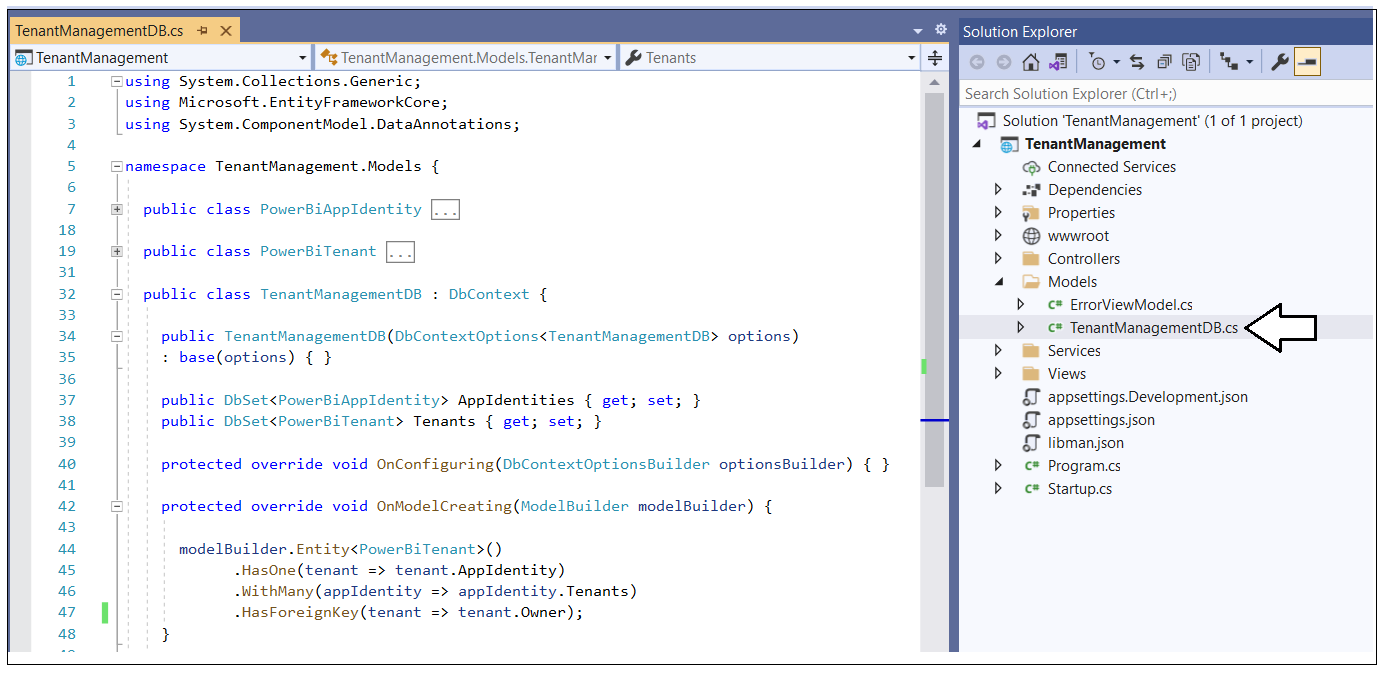


### Create the **TenantManagementDB** database

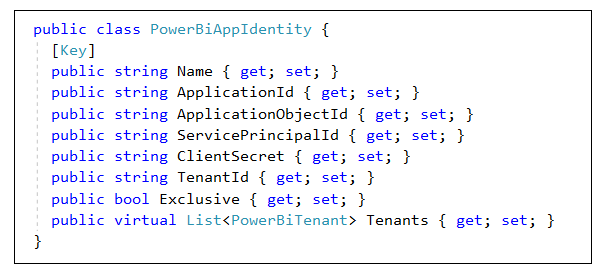
Before you can run the application in Visual Studio, you must create the project database named **TenantManagementDB**. This database schema has been created using the .NET 5 version of the Entity Framework. Creating the database will require you to run two PowerShell commands in Visual Studio.

Before creating the **TenantManagementDB** database, take a moment to understand how it’s been structured. Start by opening the file named **TenantManagementDB.cs** in the **Models** folder. Note that you shouldn't make any change to **TenantManagementDB.cs**. You are just going to inspect the file you understand how the **TenantManagementDB** database is generated.

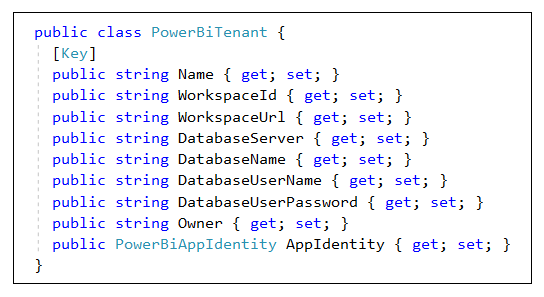
When you inspect the code inside **TenantManagementDB.cs**, you will see a class named **TenantManagementDB** that derives from **DbContext** to add support for automatic database generation using Entity Framework. The **TenantManagementDB** class serves as the top-level class for the Entity Framework which contains two **DBSet** properties named **AppIdentites** and **Tenants**. When you generate the database, each of these **DBSet** properties will be created as database tables.



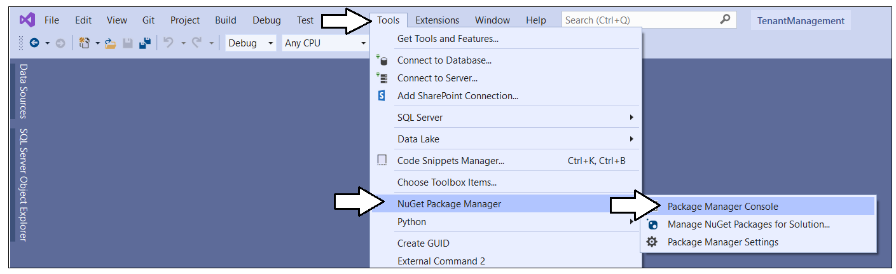
The **AppIdentites** table is generated using the table schema defined by the **PowerBiAppIdentity** class.



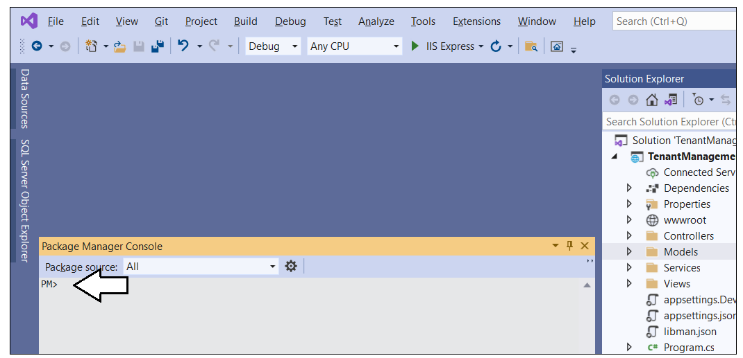
The **Tenants** table is generated using the table schema defined by the **PowerBiTenant** class.



After you have inspected the code used to generated the database, close the source file named **TenantManagementDB.cs** without saving any changes. The next step is to run the PowerShell commands to create the project database named **TenantManagementDB**. Open the Package Manager console by invoking the **Tools > NuGet Package Manager > Package Manager Console** command.



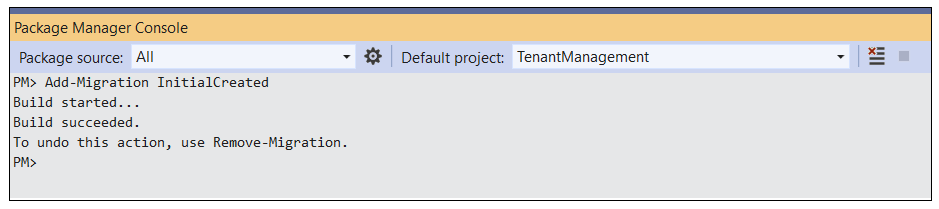
You should now see the command prompt for the **Package Manager Console** where you can type and execute PowerShell commands.



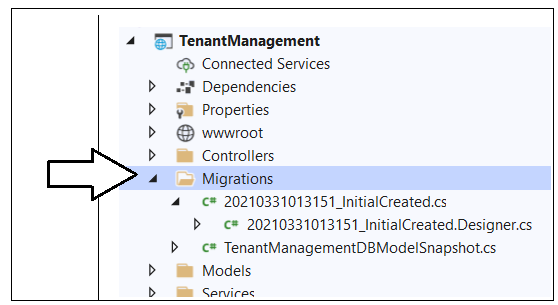
Type and execute the following **Add-Migration** command to create a new Entity Framework migration in the project.

Add-Migration InitialCreate

The **Add-Migration** command should run without errors. If it fails you might have to modify the database connection string in **appsettings.json**.



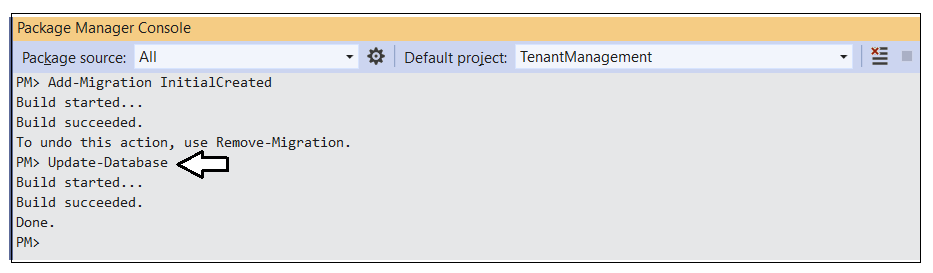
After running the Add-Migration command, you will see a new folder has been added to the project named **Migrations** with several C# source files. There is no need to change anything in thee source files but you can inspect what's inside them if you are curious how the Entity Framework does its work.



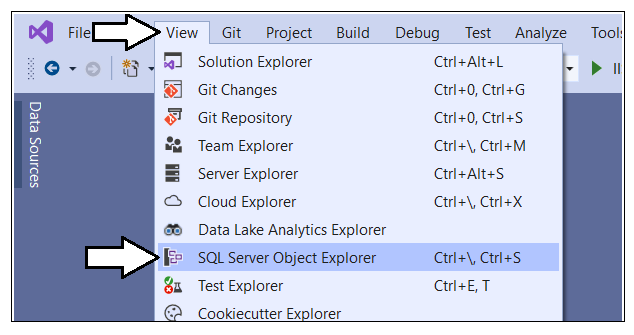
Return to the **Package Manager Console** and run the following **Update-Database** command to generate the database named **TenantManagementDB**.

Update-Database

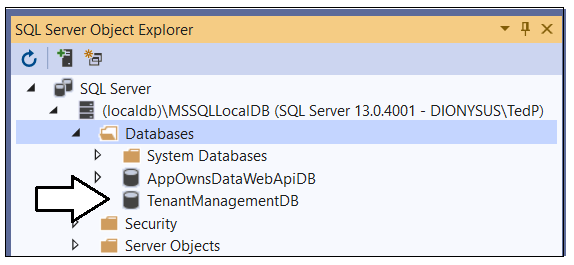
The **Update-Database** command should run without errors and generate the database named **TenantManagementDB**.



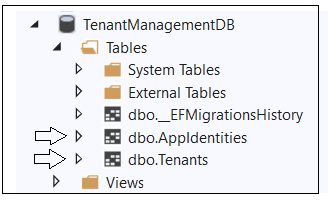
In Visual Studio, you can use the **SQL Server Object Explorer** to see the database that has just been created. Open the **SQL Server Object Explorer** by invoking the **View >** **SQL Server Object Explorer** menu command.



Expand the **Databases** node for the server you are using and verify you an see the new database named **TenantManagementDB**.



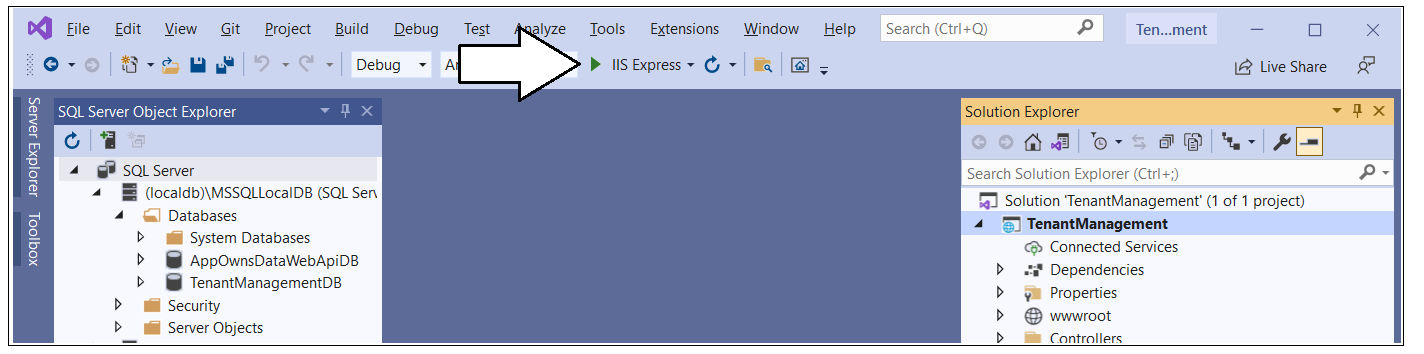
If you expand the **Tables** node for **TenantManagementDB**, you should see the two tables named **AppIdentities** and **Tenants**.



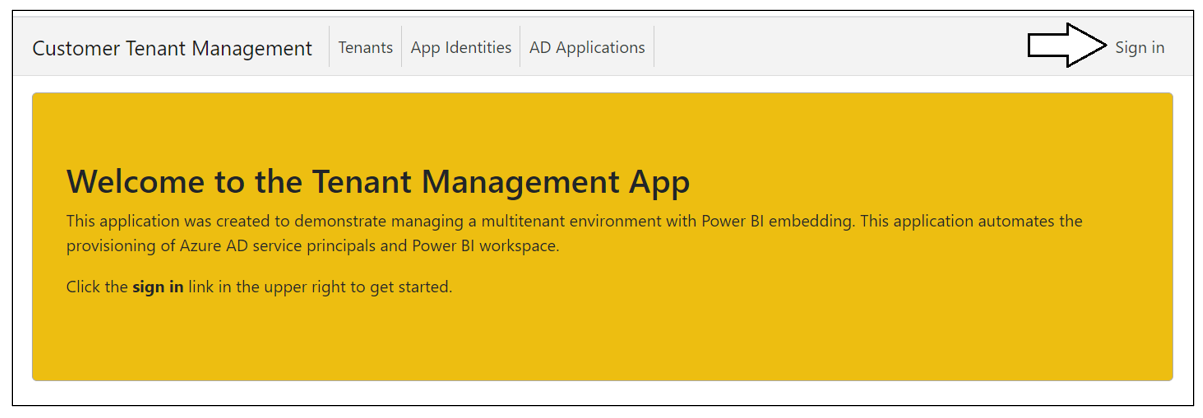
The **TenantManagementDB** database has now been set up and you are ready to run the application in the Visual Studio debugger.

## Test the Tenant Management Application

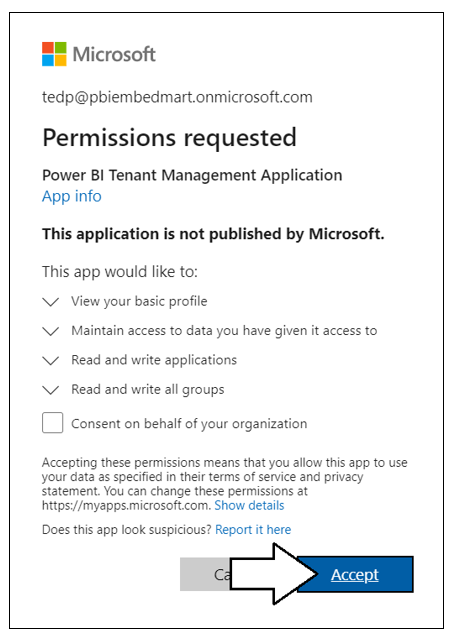
Launch the **TenantManagement** web application in the Visual Studio debugger by pressing the **{F5}** key or clicking the Visual Studio **Play** button with the green arrow and the caption of **IIS Express**.



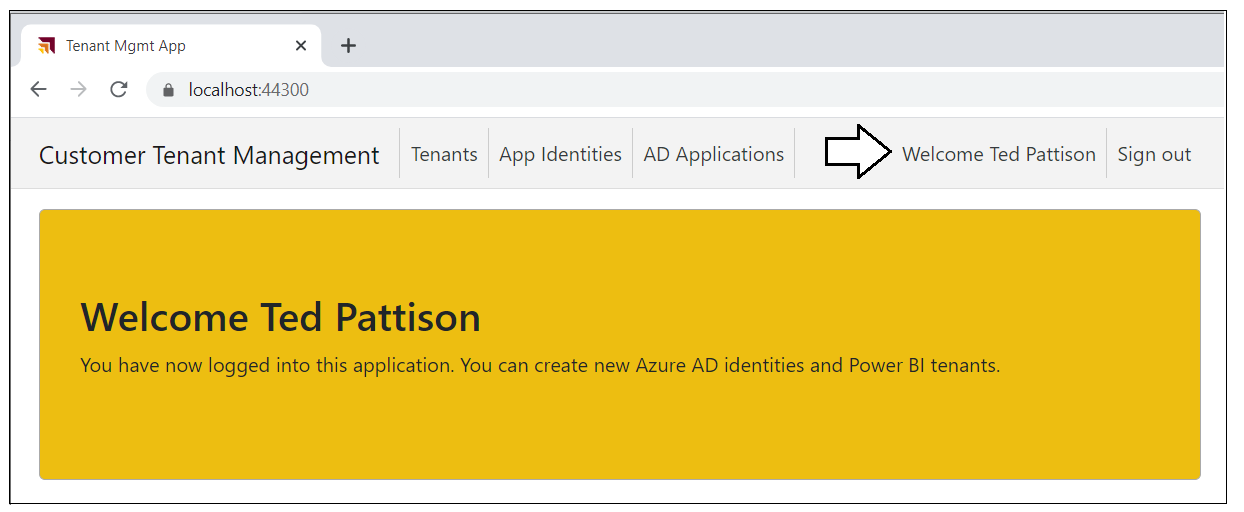
When the application starts, click the **Sign in** link in the upper right corner to begin the user login sequence.



The first time you authenticate with Azure AD, you'll be prompted with the following dialog asking you to accept the delegated permission request that the application has made for the Microsoft Graph API. Click the **Accept** button to grant these permissions and continue.



Once you have logged in, you should see your name in the welcome message.

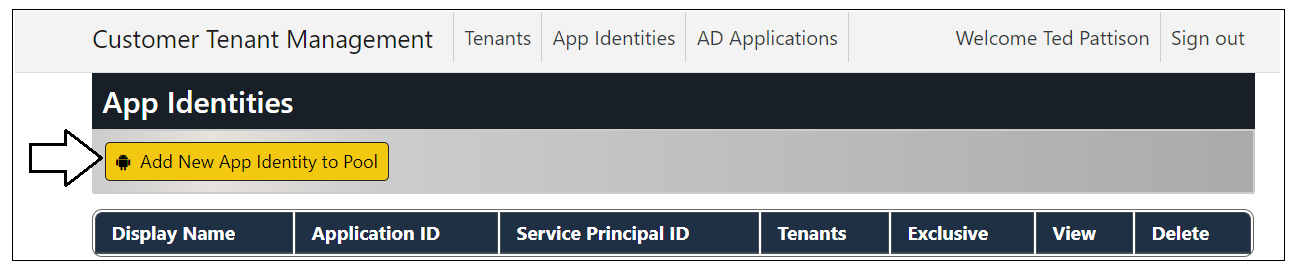


### Create App Identities

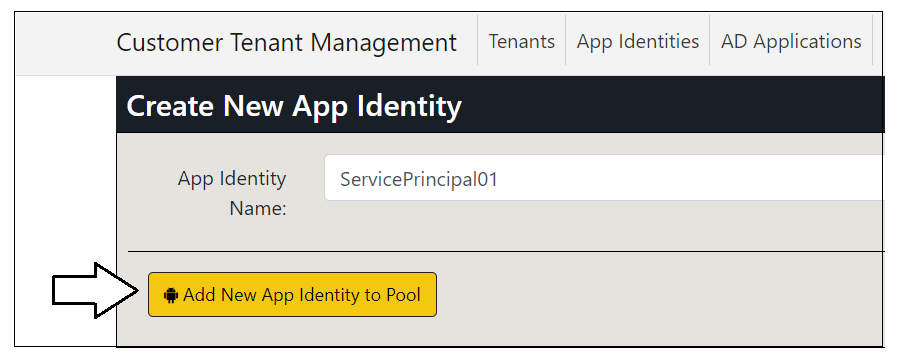
Now, you should start by creating a few new App Identities. Click the **App Identities** link to navigate to the **App Identities** page.



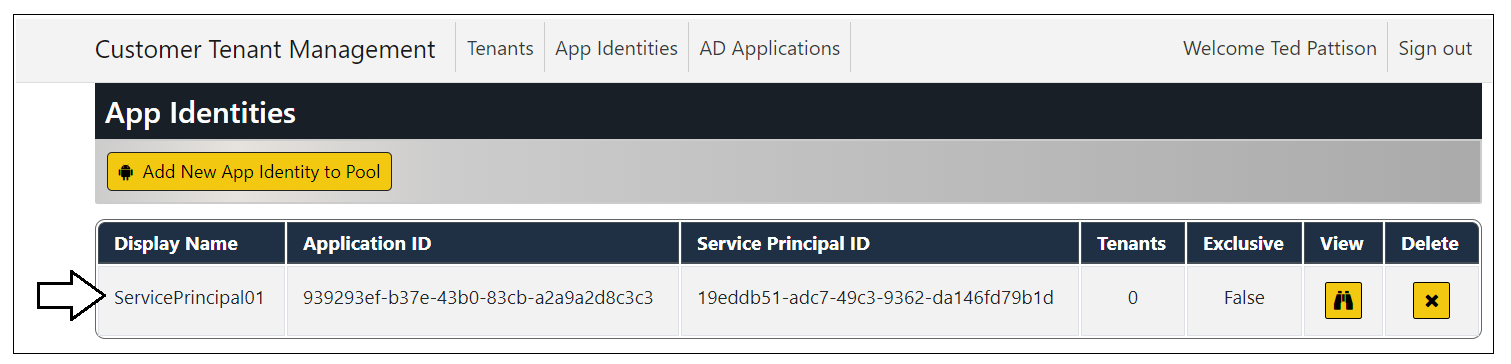
Click the **Add New App Identity to Pool** button to display the **Create New App Identity** page.



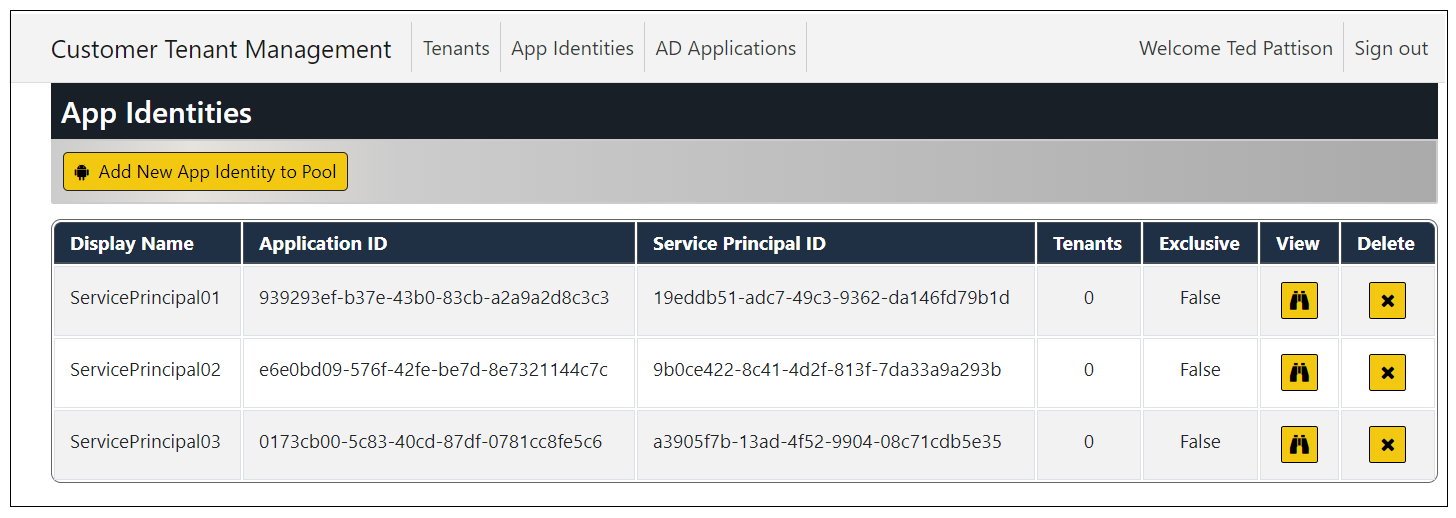
The **Create New App Identity** page will automatically populate the textbox with the name with a value of **ServicePrincipal01**. Click the **Add New App Identity to Pool** button to create the new app identity.



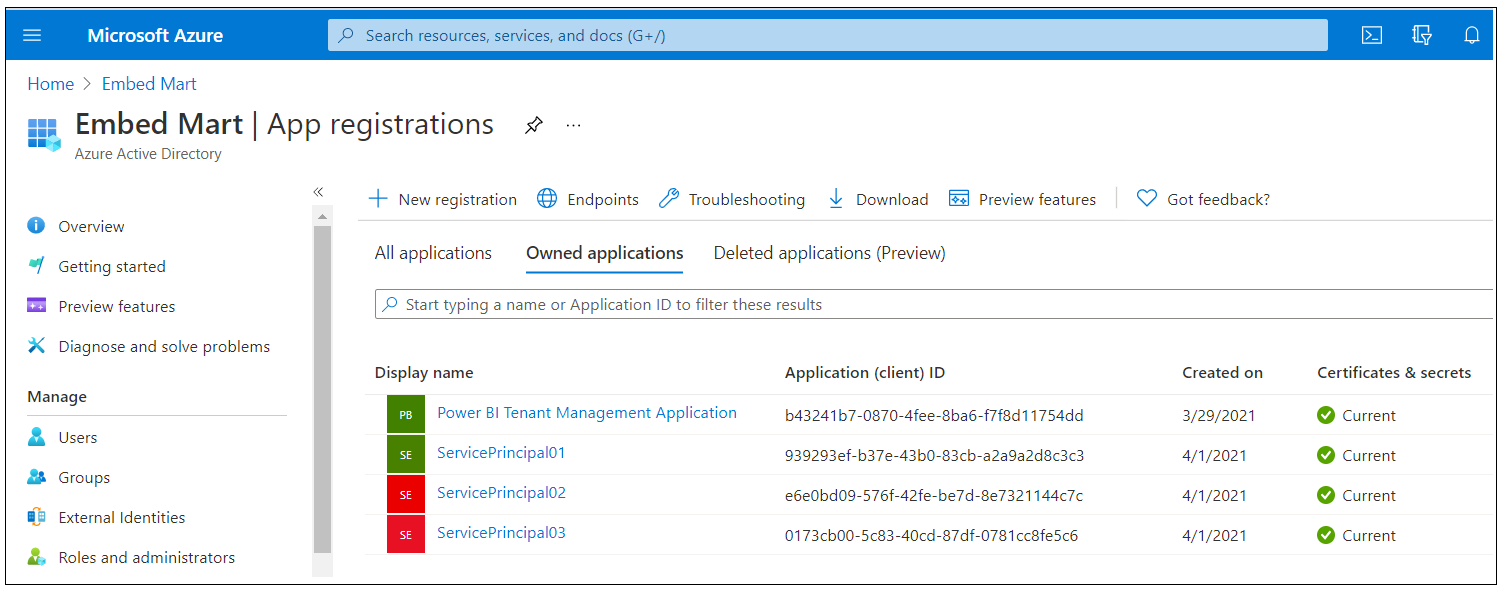
After a few second, you should see a new row in table on the **App Identities** page with **ServicePrinicpal01**.



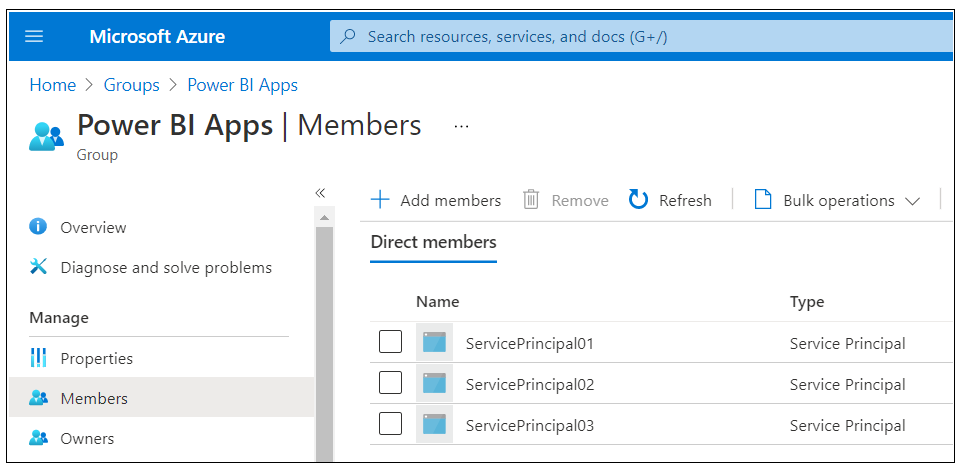
Follow the same steps to create two more app identities named **ServicePrincipal02** and **ServicePrincipal03**. When you're done, the **App Identities** page should match the following screenshot.



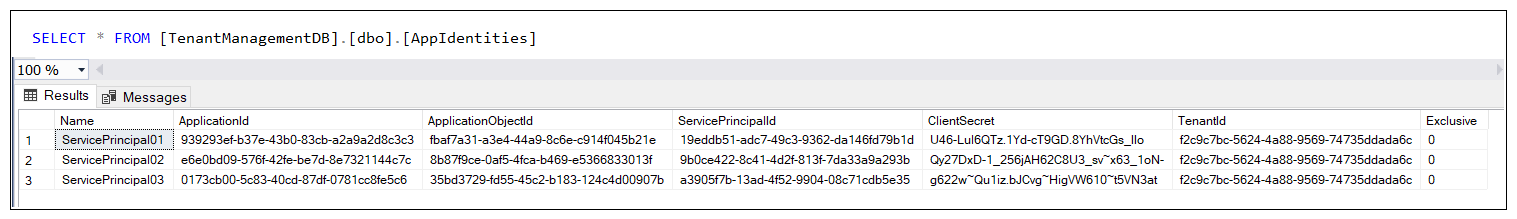
Note that behind the scenes the **TenantManagement** application is using the Microsoft Graph API to create new Azure AD application each time you create a new app identity. If you return pack to the [App registration page](https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps) in the Azure portal you will see that an Azure AD application has been created for each app identity you've created.



If you return to the [Groups](https://portal.azure.com/#blade/Microsoft_AAD_IAM/GroupsManagementMenuBlade/AllGroups) page in the Azure portal and drill into the Members page of the **Power BI Apps** security group, you will see that the **TenantManagement** application has also added the service principal for each azure AD application as a group member. This is important because these service principals must be added to this group in order to call the Power BI REST API.



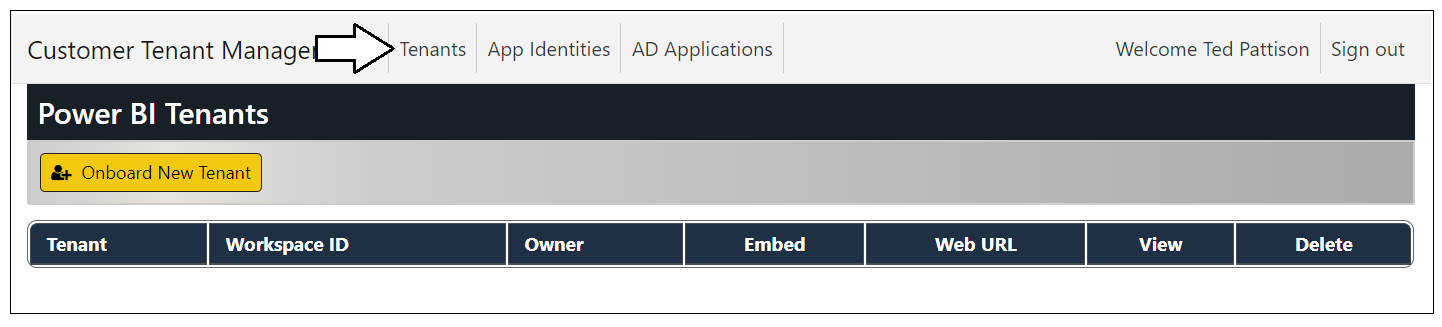
In addition to communicating with Azure AD to create and configure Azure AD application, the **TenantManagement** application also captures metadata and authentication credentials and stores them in the **TenantManagementDB** database. The **TenantManagement** application is able to retrieve these credentials and authenticate with Azure AD under the identity of any of these Azure AD applications.



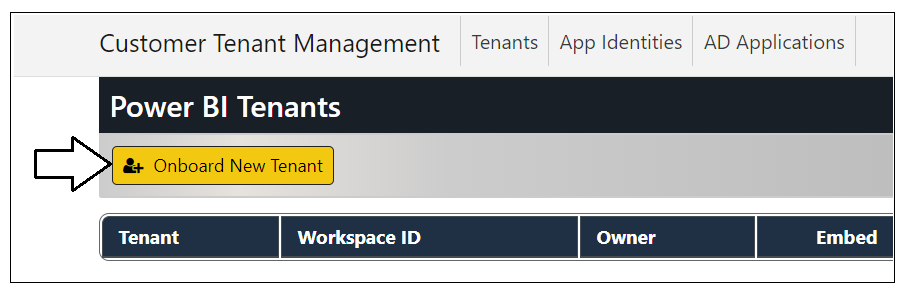
**CAVEAT**: Keep in mind that the **TenantManagement** application has been designed as a proof-of-concept (POC) application to teach concepts and provide a starting point for other developers. This application does not include certain aspects that are important to include in a real-world applications such as hiding secrets. If you plan to extend this POC sample application into a production application, it will be your responsibility to add support for hiding credentials such as the Client Secret. You can consider an approach such as using the [Always Encrypted](https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine?view=sql-server-ver15) feature in Azure SQL or extending the **TenantManagement** application to store client secrets or client certificates in [Azure Key Vault](https://docs.microsoft.com/en-us/azure/key-vault/general/basic-concepts).

### Create New Tenants

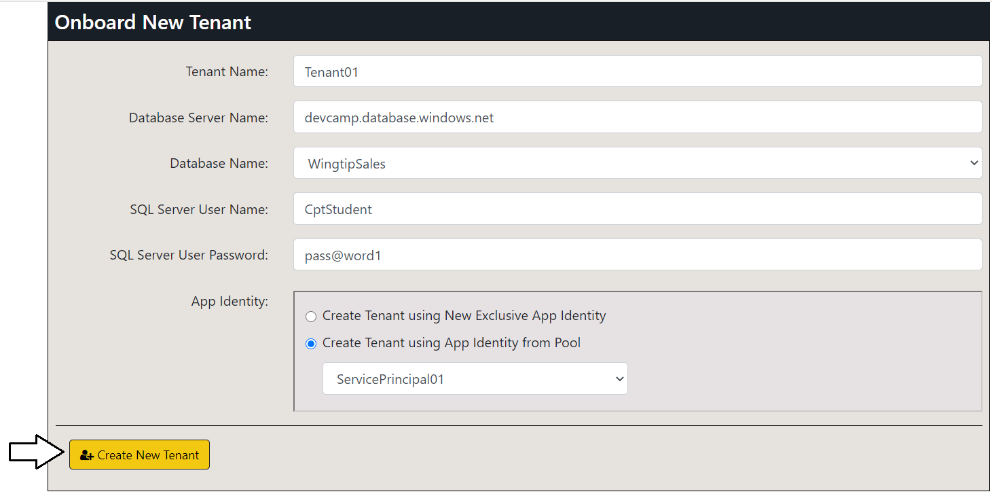
Return to the **TenantManagement** application and navigate to the **Tenants** page.



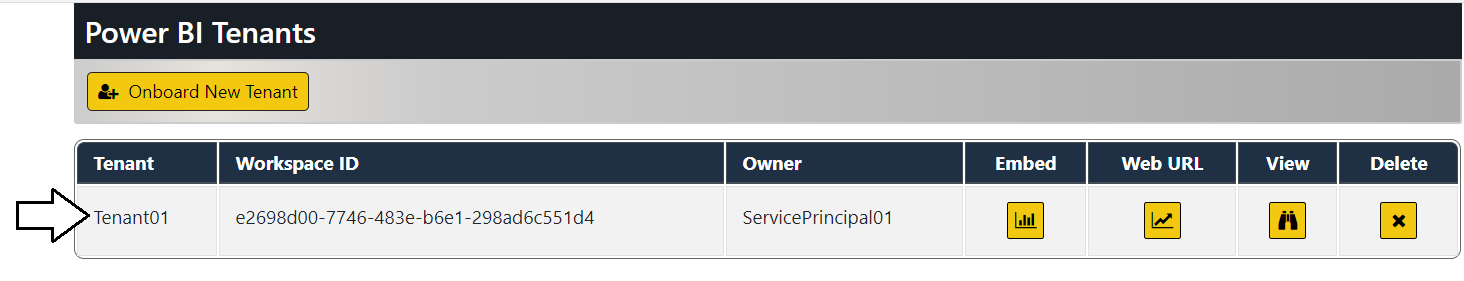
Click the **Onboard New Tenant** button to display the **Onboard New Tenant** page.



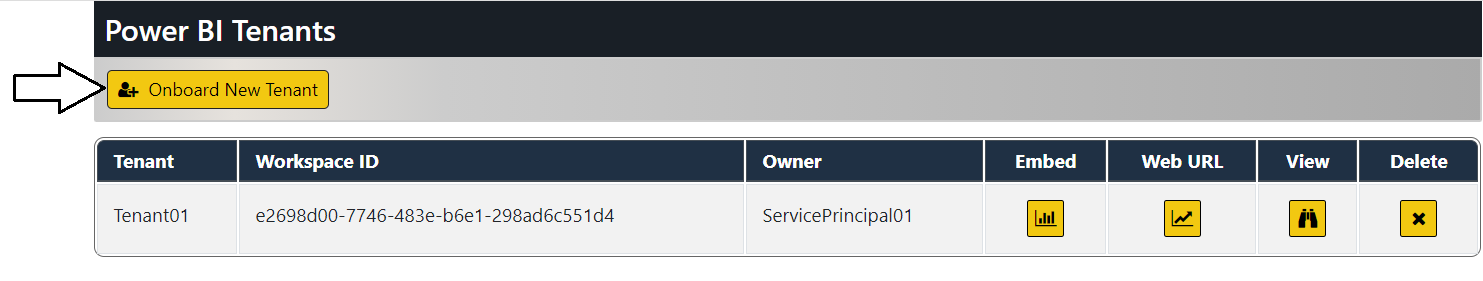
You can create the first tenant using the default values supplied by the Onboard New Tenant page. Click to **Create New Tenant** button to begin the process of creating a new customer tenant.



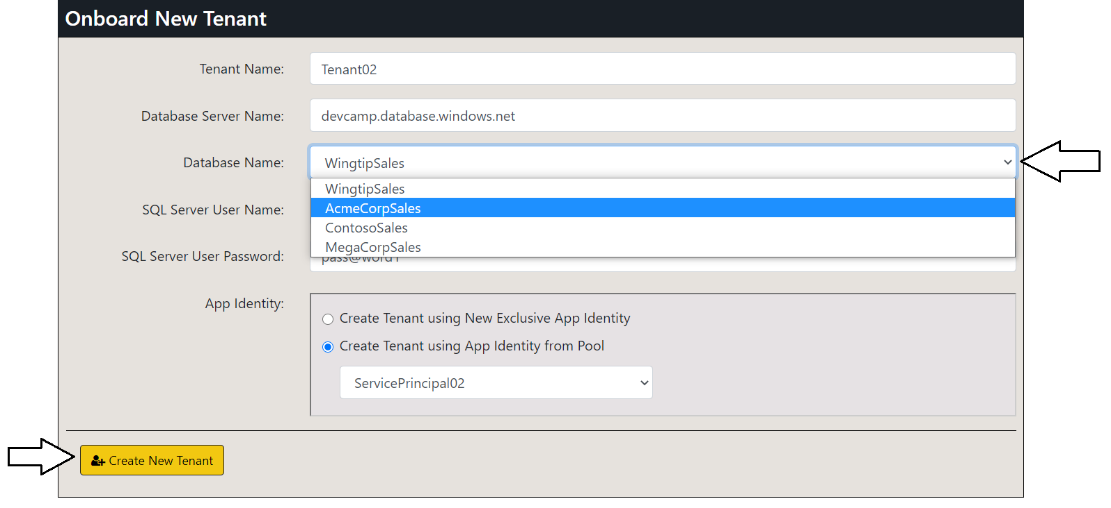
After a few seconds, you should see the next tenant has been created.



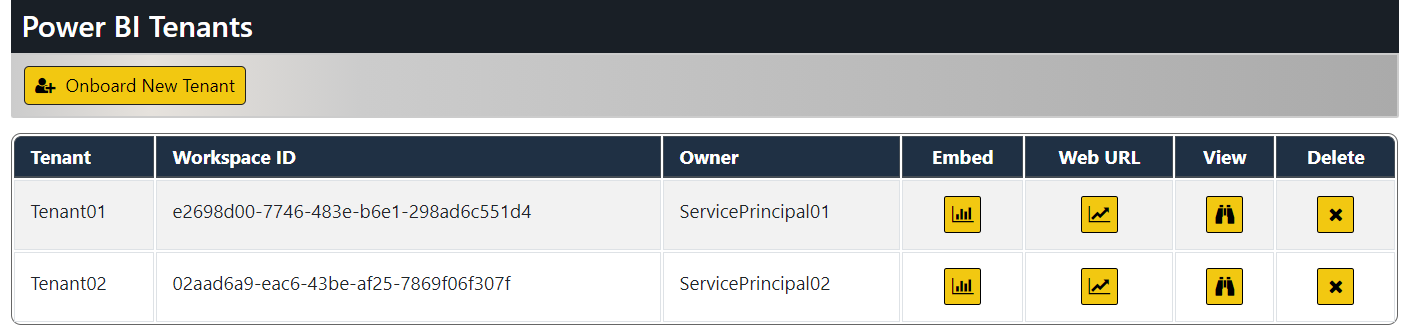
Click the **Onboard New Tenant** button again to create a second tenant.



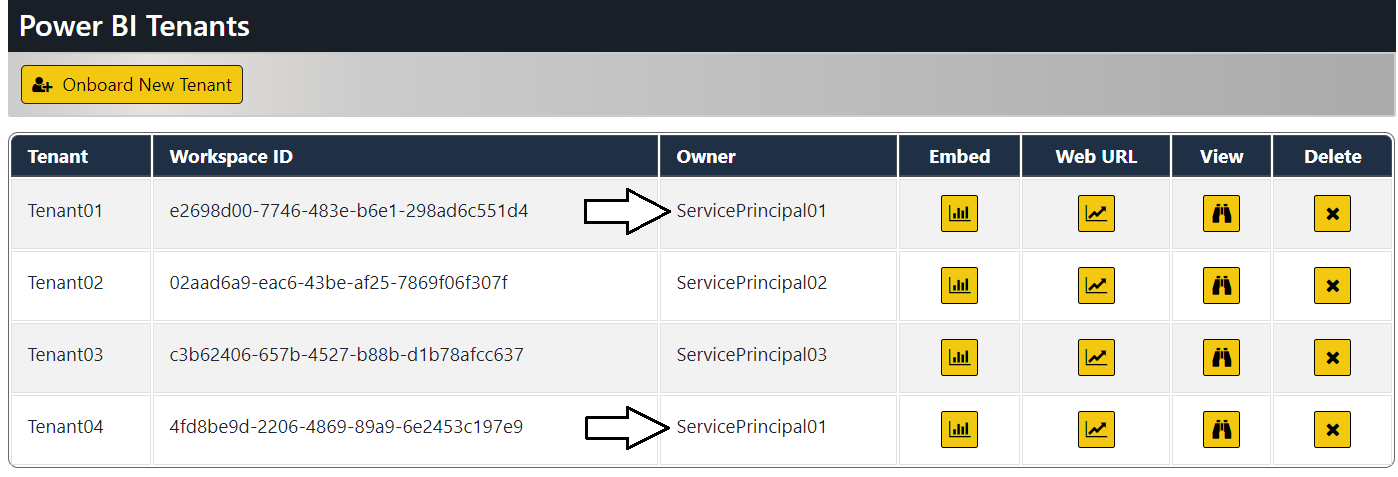
When creating the second customer tenant, select a different database than the default selection and then click the **Create New Tenant** button.



You should now have two customer tenants. Note they each tenant has a different app identity as its **Owner**.



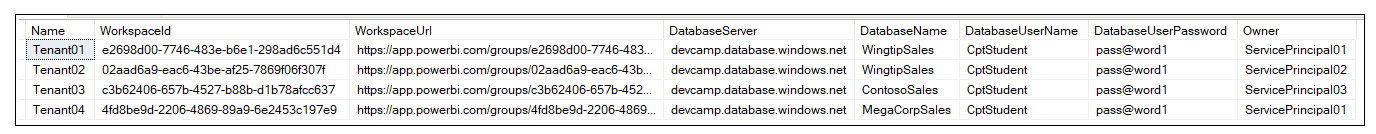
Follow the same steps to create two more customer tenants so that there are 3 app identities and 4 customer tenants. Once you have created more tenants then app identities, you should see app identity pooling where multiple customer tenants share the same app identity.



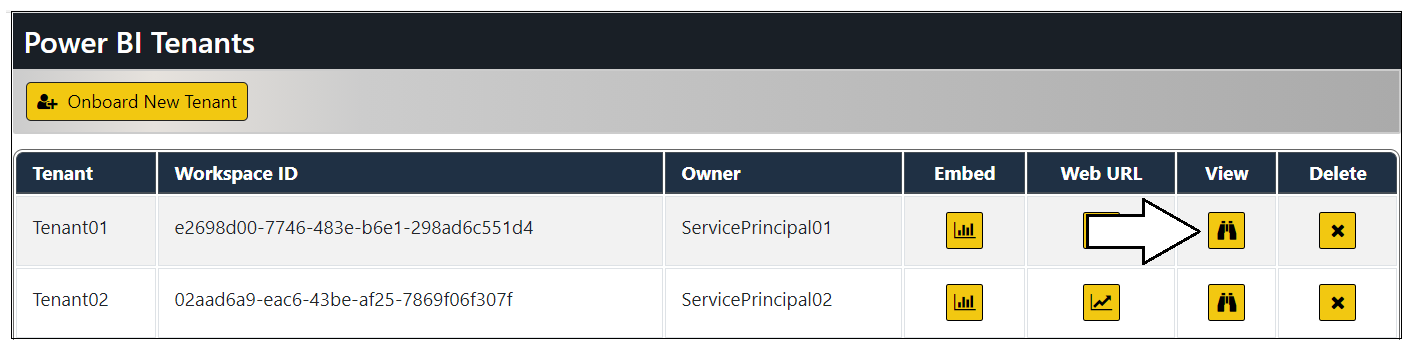
As you create a new customer tenant, the **TenantManagement** application uses the Power BI REST API to implement the onboarding logic.

1. Create a new Power BI workspace
2. Upload a template PBIX file to create **Sales** dataset and **Sales** report
3. Update dataset parameters on **Sales** dataset to point to this customer's database
4. Patch credentials for the SQL datasource used by the **Sales** dataset
5. Start a refresh operation on the **Sales** database

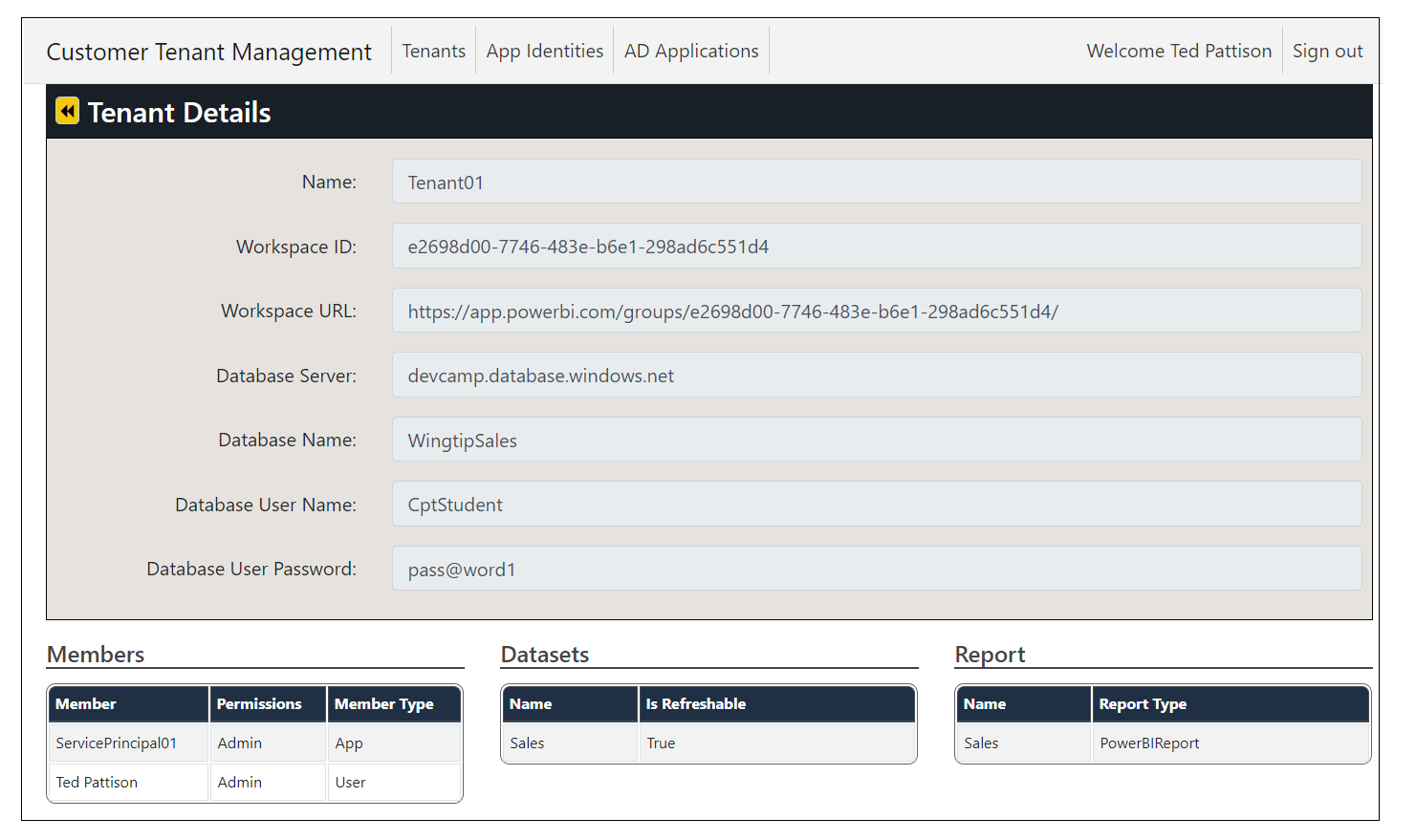
The **TenantManagement** application also create a new record in the **Tenants** table of the **TenantManagementDB** database. Note that the application identity associated with this customer tenant is tracked in the **Owner** column.



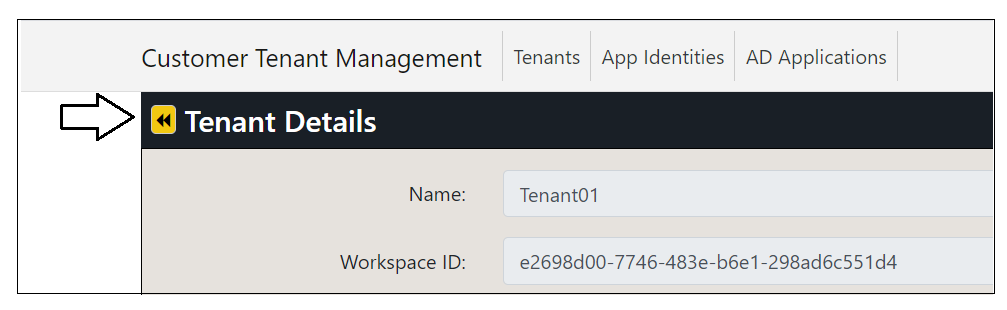
You can click on the **View** button for a specific tenant on the **Power BI Tenant** page to drill into the **Tenant Details** page.



The bottom of the **Tenant Details** page also shows details of the underlying Power BI workspace including its members, datasets and reports.

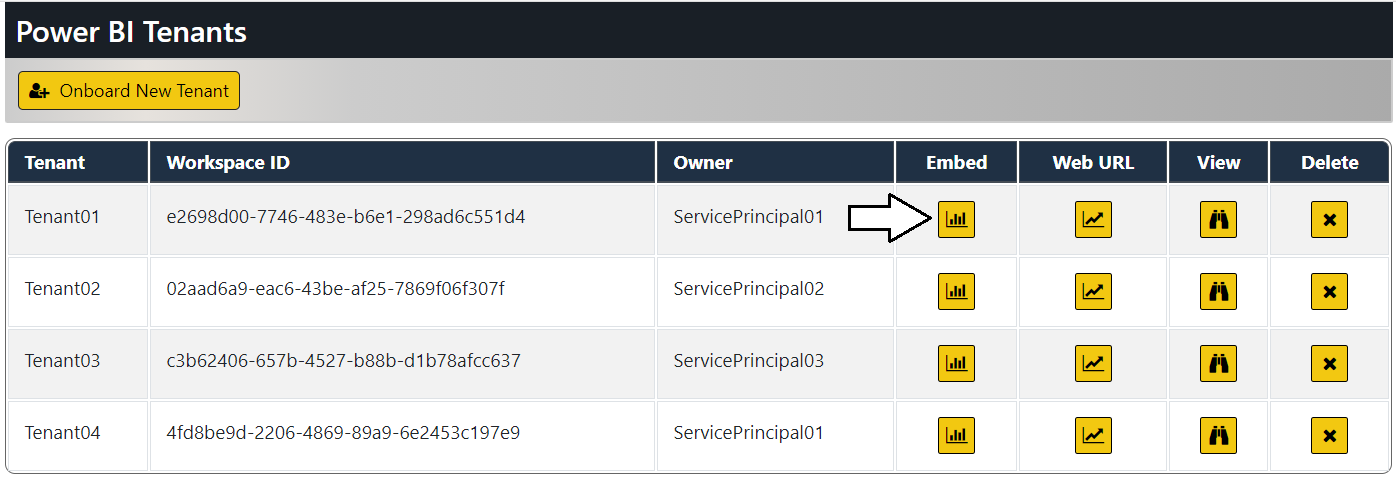


Click on the back arrow to return to the Power BI Tenants page.



### Embed Reports

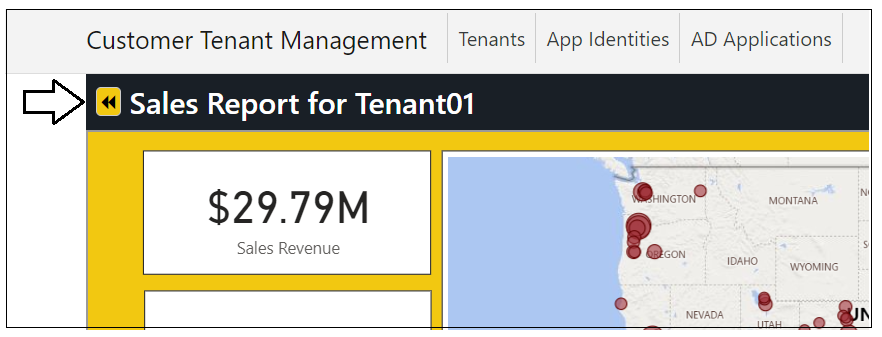
Now it's time to make use of the **TenantManagement** application's ability to embed reports. Click on the **Embed** button for the first customer tenant.



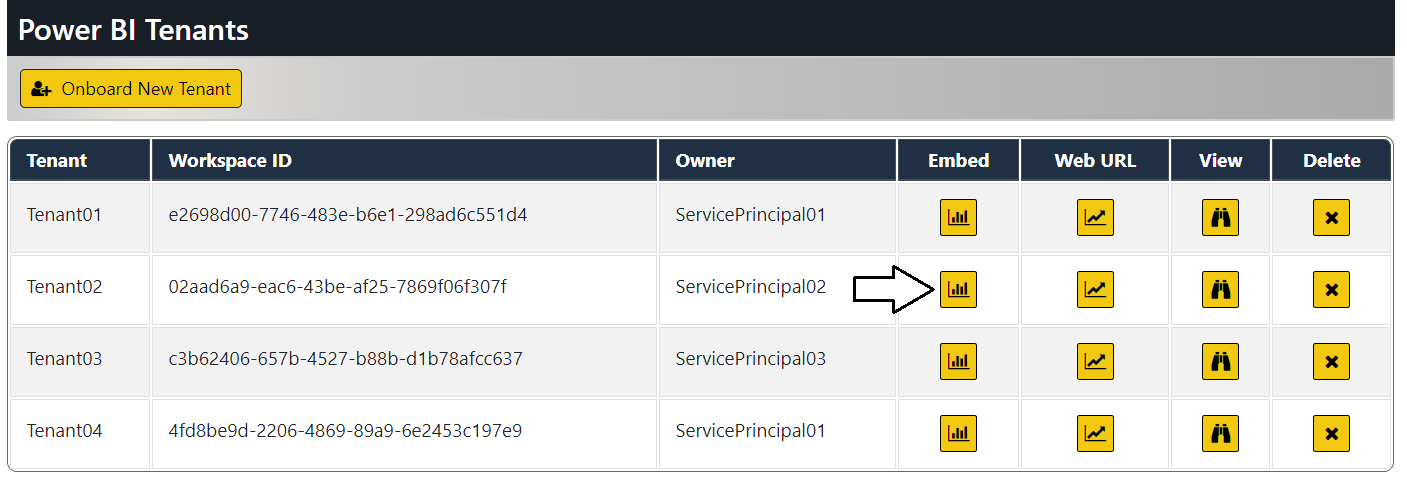
You should see a page with an embedded report for that tenant. When you click on this button to embed a report, the **TenanantManagement** application retrieves credentials for the app identity associated with the customer tenant and uses Client Credentials Flow to acquire an app-only access token from Azure AD. That access token is then used to communicate with the Power BI Service to retrieve report metadata and generate an embed token for the embedding process.



Click on the back arrow button to return to the **Tenants** page.

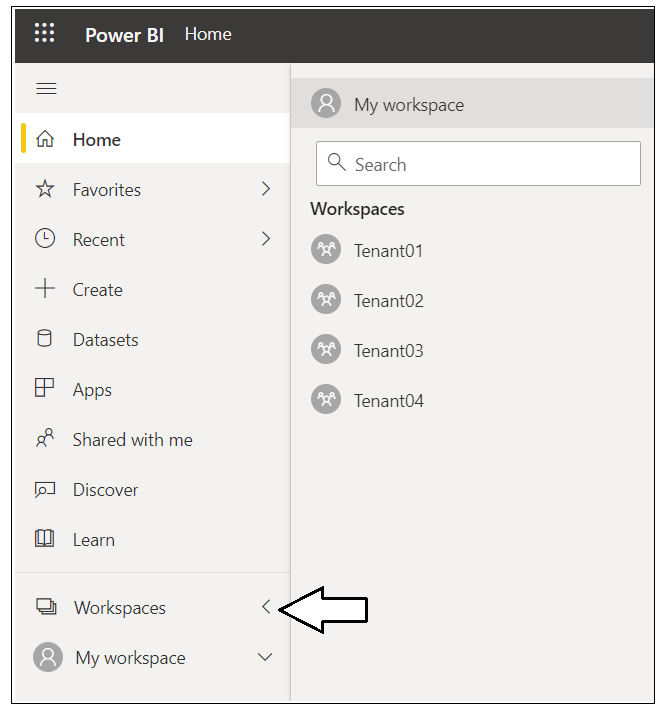


Now test clicking the **Embed** button for other customer tenants. As you can see, the **TenantManagement** application has the ability to generate access tokens for any of the Azure applications that it has created.

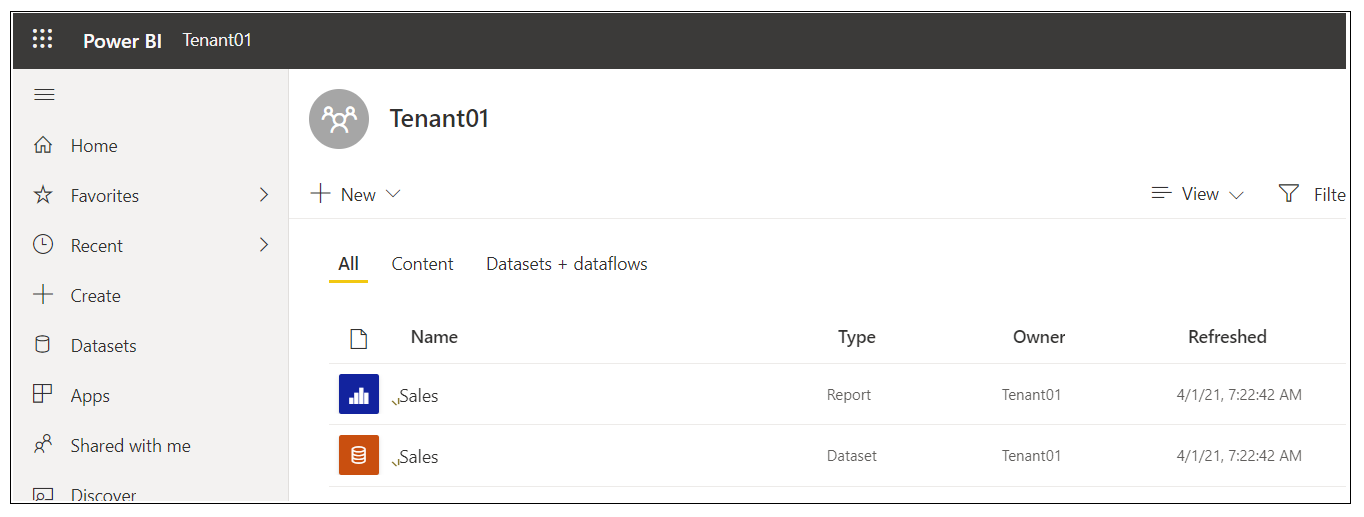


Inspect the Power BI Workspaces

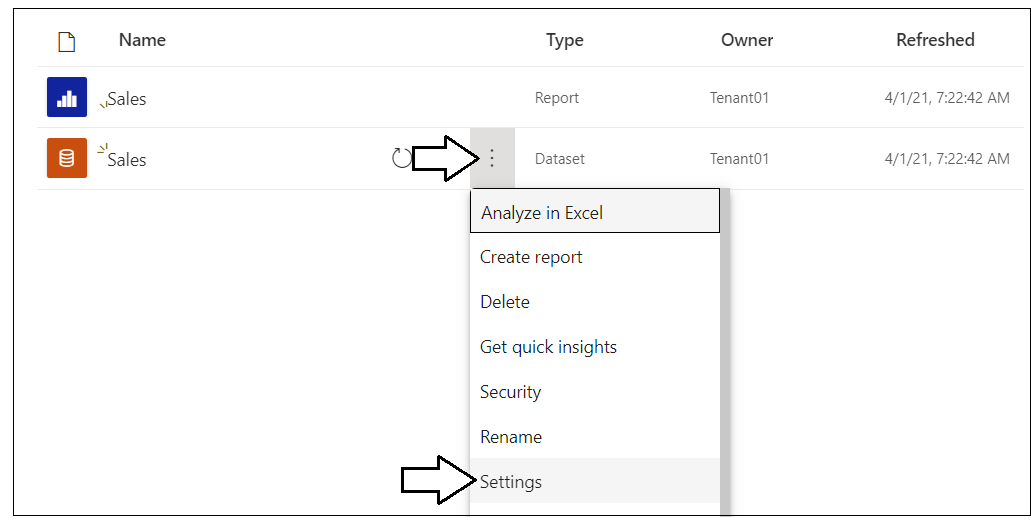
If you're curious about what's been created in Power BI, you should be able to go to the Power BI Service and examine the workspaces. Navigate to the Power BI Service portal at <https://app.powerbi.com> and examine the workspaces that have been created.



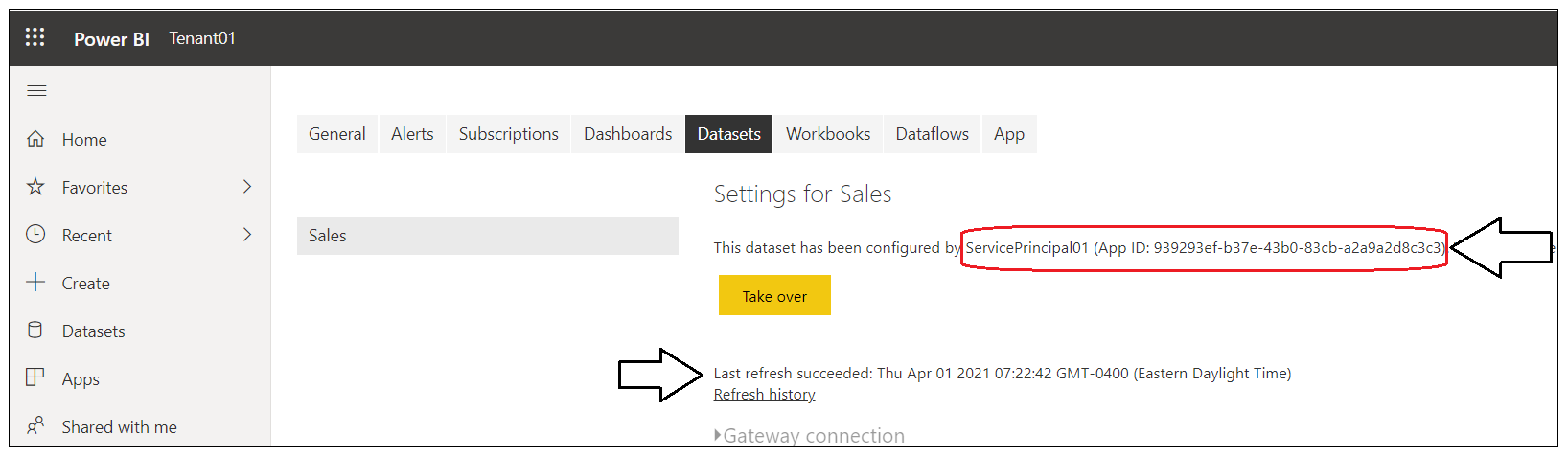
Navigate to one of these workspaces such as **Tenant01**.



Drill into the Setting page of the dataset named **Sales**.



You should be able to verify that the dataset has been configured by one of the Azure AD application created by the **TenantManagement** application.



This concludes the walkthrough of the **TenantManagement** application.