

Fundraising and Engagement

for Dynamics 365 Sales, a Microsoft solution built with MISSION CRM

Deployment Guide

**Published**: October 9, 2020   
**Release Version**: v1.0.0.2

Contents

[Overview 2](#_Toc56187805)

[Getting Started 3](#_Toc56187806)

[Intended Audience 3](#_Toc56187807)

[Prerequisites 3](#_Toc56187808)

[Configuring Your Application Registration in Dynamics 11](#_Toc56187809)

[Deploying Fundraising and Engagement for Dynamics 365 Sales Enterprise 12](#_Toc56187810)

[Prerequisites Installed During Solution Deployment 12](#_Toc56187811)

[Acquire Fundraising and Engagement from within Dynamics 13](#_Toc56187812)

[Acquire from AppSource 14](#_Toc56187813)

[Provisioning the ARM Template 16](#_Toc56187814)

[Azure Components Created 16](#_Toc56187815)

[Setting up Azure Components to Support Multiple Dynamics Environments 17](#_Toc56187816)

[Deploying the ARM Template 17](#_Toc56187817)

[Adding the Azure SQL User Accounts 21](#_Toc56187818)

[Deploying the Fundraising and Engagement components to Azure 23](#_Toc56187819)

[Clone the Solution 23](#_Toc56187820)

[Build the Solution 25](#_Toc56187821)

[Deploy to Azure 26](#_Toc56187822)

[Populate the Azure SQL Schema (Using C# Migrations) 33](#_Toc56187823)

[Post Deployment Tasks (Dynamics 365) 38](#_Toc56187824)

[Configuration Record Prerequisites 38](#_Toc56187825)

[Assigning Configuration Settings to Dynamics 365 Users 41](#_Toc56187826)

[Troubleshooting 43](#_Toc56187827)

[When a Failure Occurs 43](#_Toc56187828)

[Updating Visual Studio 2019 Community Edition (or Higher) 43](#_Toc56187829)

[Modifying the Template 43](#_Toc56187830)

[Naming and Unique Name Concepts 43](#_Toc56187831)

[Hitting Maximum Path Length Limit When Deploying Azure Components 44](#_Toc56187832)

[Learn More About Azure and Extending Dynamics 365 44](#_Toc56187833)

# Overview

Fundraising and Engagement built for Dynamics 365 Sales, a Microsoft solution built with MISSION CRM, requires an accompanying single tenant Azure environment to access features such as recurring donation processing, Bank ACH batching, and automated financial summaries. Although this functionality resides in the Fundraising and Engagement Dynamics 365 solution, the solution relies on an Azure environment to process, generate, and calculate data on an ongoing basis.

To set up an Azure environment for a Fundraising and Engagement instance, an Azure [ARM template](https://aka.ms/fe-armtemplate) has been pre-configured to allow implementors to provision the required environment with as little intervention as possible.

This guide walks an implementor through the process of deploying the Azure ARM template and ensuring it is correctly configured. The process assumes that the user deploying the Azure ARM template thoroughly understands the prerequisites listed on page 4 prior to completing this process.

# Getting Started

This guide is intended to help implementors navigate the initial setup of a new Azure environment that is supporting a new instance of Dynamics 365 Customer Engagement which has already installed and configured the Fundraising and Engagement managed solution. Take a moment to review the requirements prior to moving ahead. The process requires that the user take note of the results of the creation of the Azure environment as they are needed to complete the setup process.

## Intended Audience

This guide assumes the user implementing the Fundraising and Engagement Azure environment has a solid understanding of Azure, Web Applications, and has completed AZ-900 or comparable accreditation at a minimum.

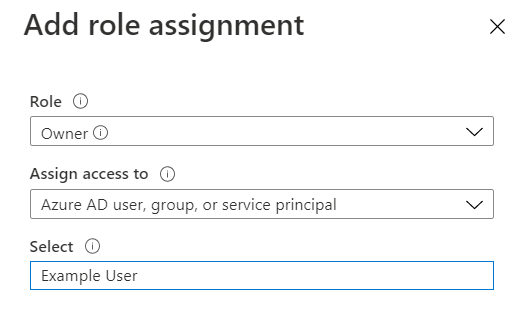
## Prerequisites

The following prerequisites are required to deploy the template and create the Azure components.

|  |
| --- |
| Note: The ARM template is not capable of rolling back components. This means if the ARM template fails as a result of the following prerequisites not being met, the user implementing the template must manually remove the created components before reattempting the deployment. |

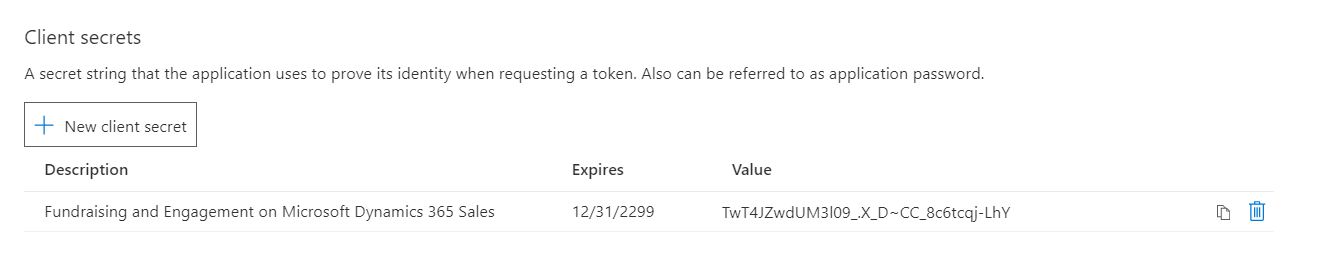
### Azure Role

The template has been designed to run as a user that has the ‘Owner’ role assignment on the target subscription (see [Azure role assignments](https://docs.microsoft.com/en-us/azure/role-based-access-control/role-assignments-portal)). During the creation process, several steps assign access privileges and associate roles to application user accounts. Without the ‘Owner’ role, the ability to manage this process automatically via the template cannot occur.



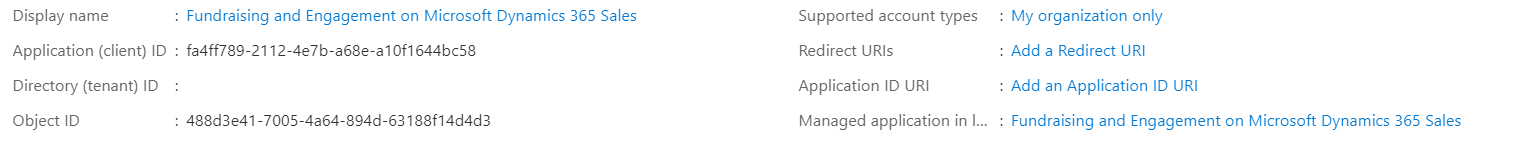
### Azure Active Directory Application Registration

The Azure Active Directory Application user must be set up and configured to have the Dynamics 365 Customer Engagement rights within the tenant. Once the Application user is set up, record the secret and client application identifier (see [Client Secrets](https://docs.microsoft.com/en-us/azure/active-directory/develop/quickstart-register-app#add-a-client-secret) in Azure documentation).



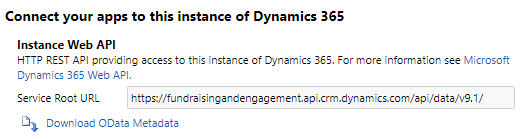
|  |
| --- |
| Note: The application client secret, once generated, cannot be retrieved after it has been configured. Be sure to record the value as it is needed in the completion of the template. The value must match or the Azure environment will not work as expected. |

The ‘Application (client) ID’ will also be required. Record the value to complete the ARM template process.



### Dynamics 365 Customer Engagement Instance Information

The Azure components configuration requires the Dynamics Customer Engagement API URL. You can find it in your Dynamics environment in settings under Advanced Settings -> Customizations -> Developer Resources.



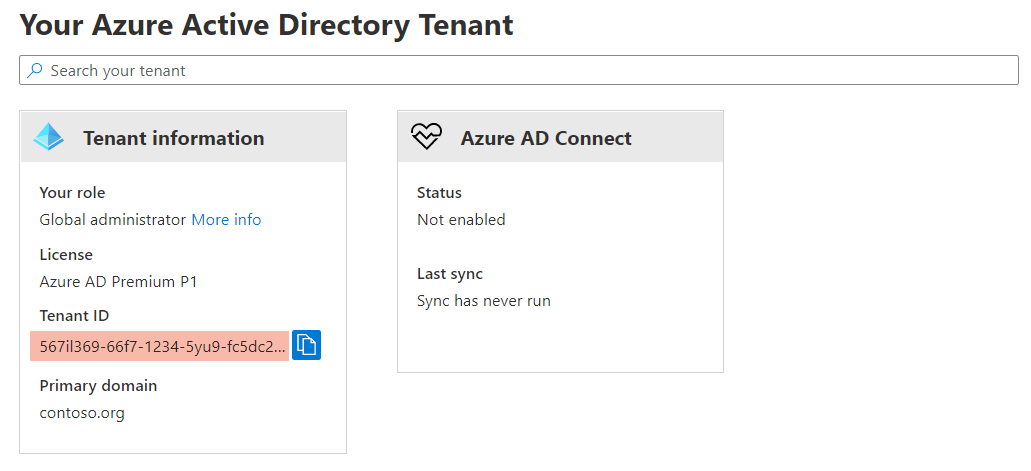
The value required will be in the format of <https://instancenameasdisplayed.api.crmx.dynamics.com>.

### Azure and Dynamics Tenant

The Azure environment you are creating the components in must be within the same Office 365 Azure Active Directory controlled tenant. Due to the Azure environment relying on server to server authentication by means of the application record in Azure Active Directory, the user must be visible to your Fundraising and Engagement for Dynamics 365 Sales solution instance.

### Retrieve Your Azure Directory Tenant ID

The Azure Active Directory Tenant Identifier is required during the setup of the Azure components. For the server-to-server authentication to work as expected, both the application identifier and the tenant identifier are required. The tenant identifier in Azure can be found by navigating to ‘Azure Active Directory’ and viewing the ‘Tenant ID’ from the overview pane.



### Configure SQL Server Management Studio (SSMS)

It is required to access the Azure SQL database that is created post completion of the ARM template deployment. Get the latest version of [SQL Server Management Studio (SSMS)](https://docs.microsoft.com/en-us/sql/ssms).

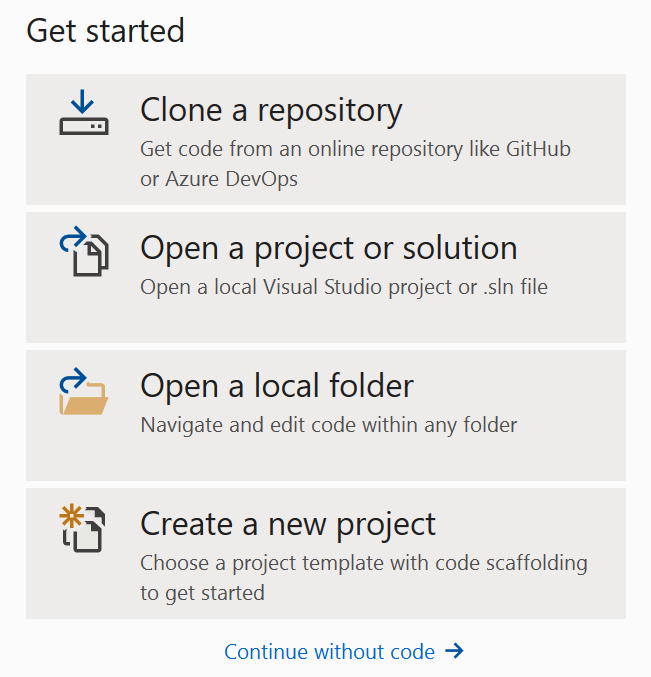
At the time of authoring this document, SQL Server Management Studio is the only SQL scripting tool that can add and provision SQL users as required by the deployment.

### Generating a Padlock token (passcode)

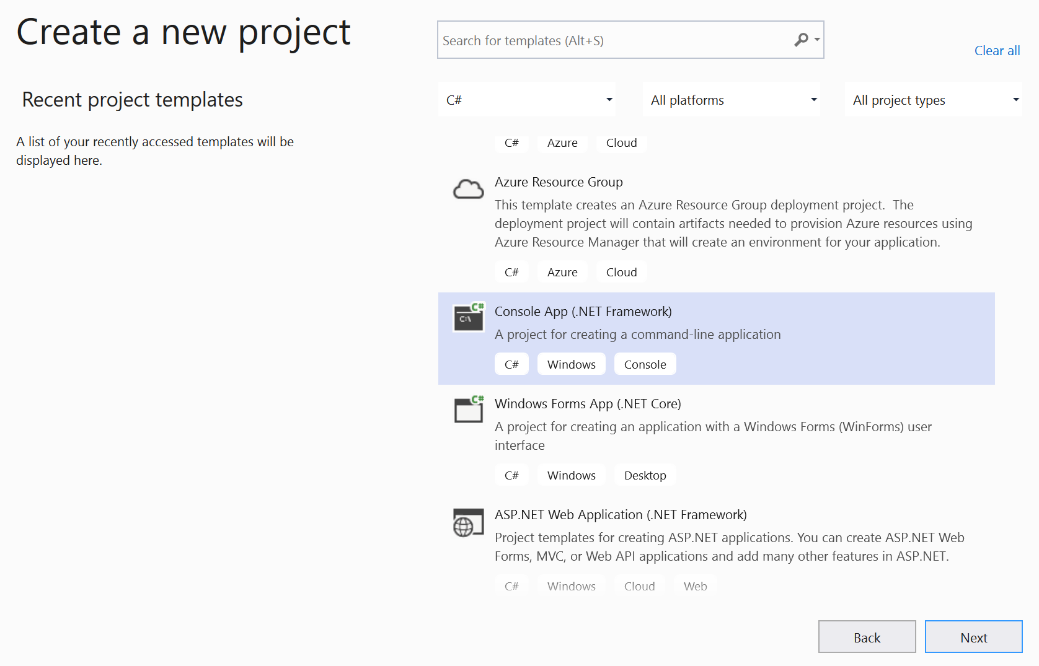
To securely connect your Dynamics 365 environment to the Azure API a client key and passcode (padlock token) are required.

To generate a new passcode, you will create a simple C# console project that takes an unencrypted passcode and generates an encrypted passcode that needs to be inserted as a configuration setting into an Azure Web Application later.

Open Visual Studio 2019 Community Edition or higher, then elect “Create a new project” from the welcome screen:

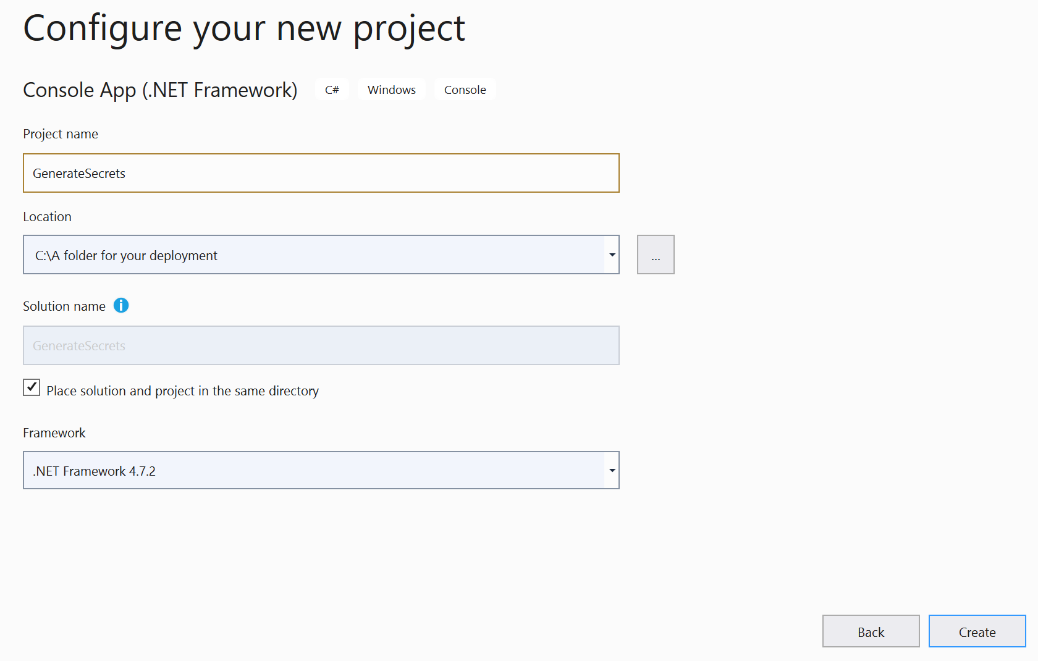


From the list of available project types, ensure “Console App (.NET Framework)” is selected, then select ‘Next’:

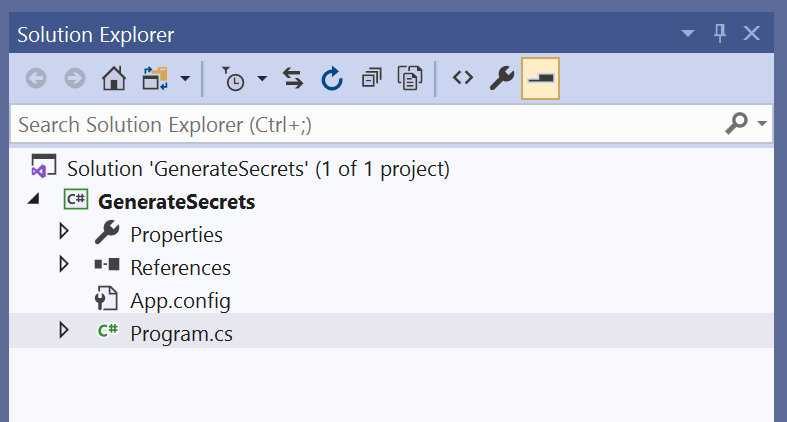


Enter in the details of your project, Project Name is the only required input. The available options are:

|  |  |
| --- | --- |
| Setting | Example Value |
| Project name | GenerateSecrets |
| Location | C:\username\Document\VisualSudio |
| Solution name | Leave as the default |
| Framework | Always set to .NET Framework 4.7.2 |



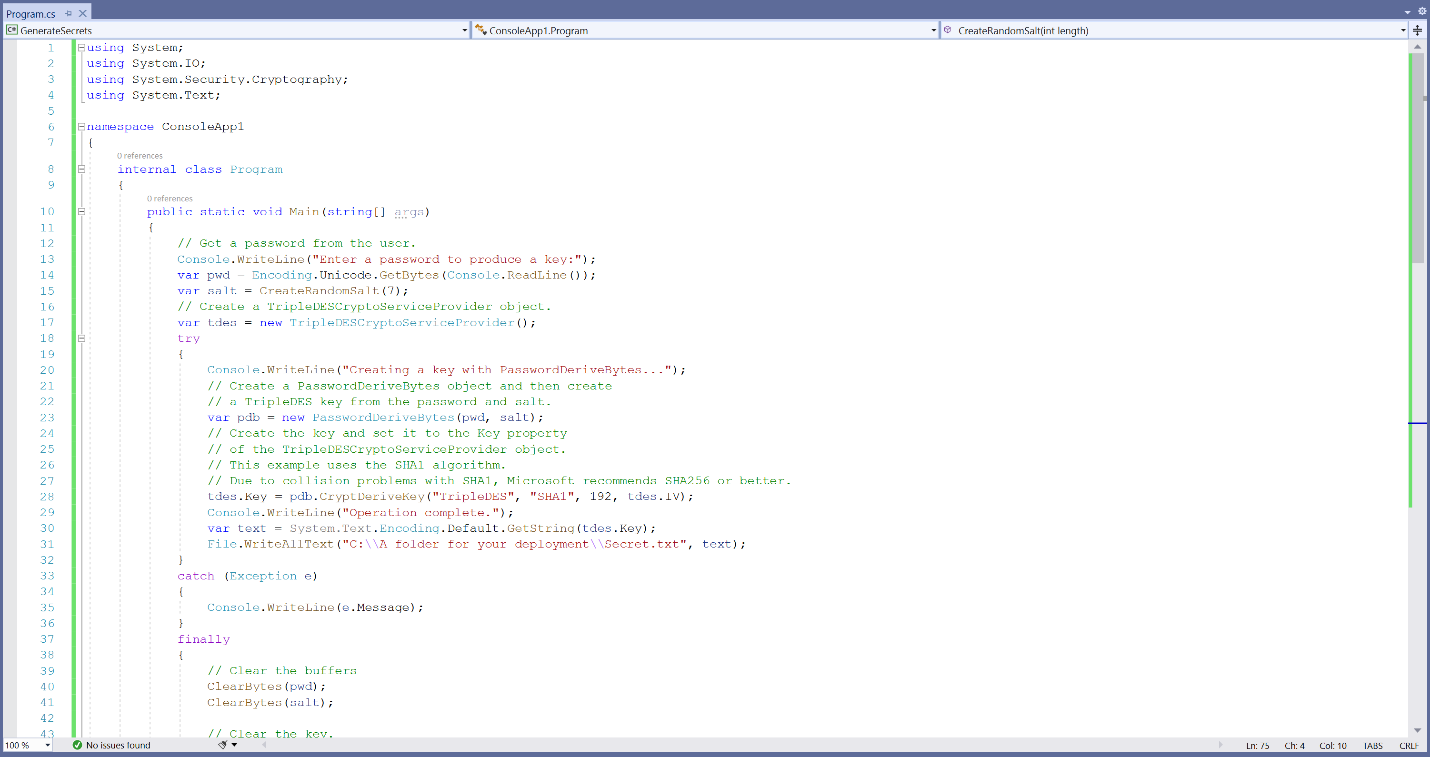
With the project created, double click on the “Program.cs” file in solution explorer:



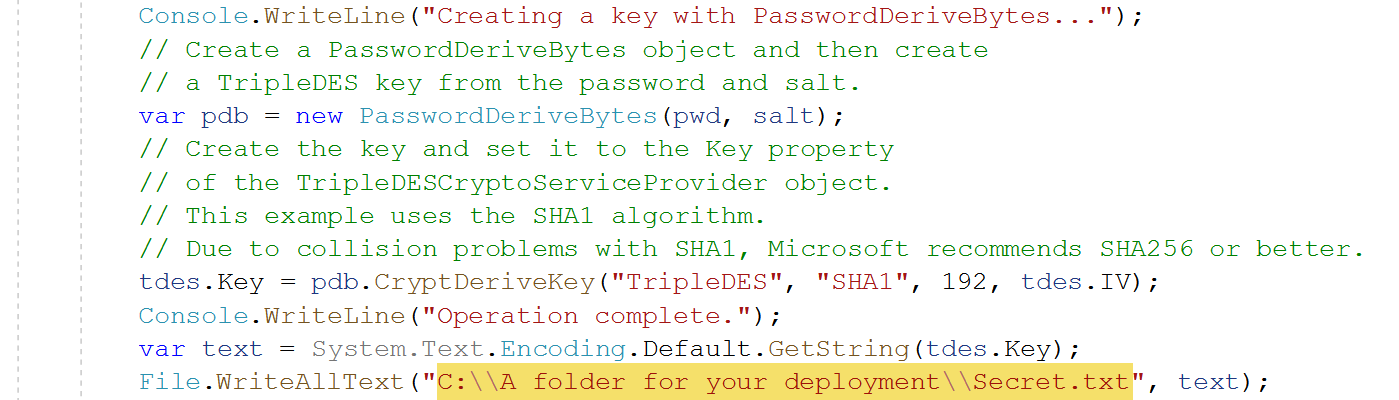
Copy the following contents (this content below should be the only content that exists in your “Program.cs” file:

|  |
| --- |
| using System;  using System.IO;  using System.Security.Cryptography;  using System.Text;  namespace ConsoleApp1  {  internal class Program  {  public static void Main(string[] args)  {  // Get a password from the user.  Console.WriteLine("Enter a password to produce a key:");  var pwd = Encoding.Unicode.GetBytes(Console.ReadLine());  var salt = CreateRandomSalt(7);  // Create a TripleDESCryptoServiceProvider object.  var tdes = new TripleDESCryptoServiceProvider();  try  {  Console.WriteLine("Creating a key with PasswordDeriveBytes...");  // Create a PasswordDeriveBytes object and then create  // a TripleDES key from the password and salt.  var pdb = new PasswordDeriveBytes(pwd, salt);  // Create the key and set it to the Key property  // of the TripleDESCryptoServiceProvider object.  // This example uses the SHA1 algorithm.  // Due to collision problems with SHA1, Microsoft recommends SHA256 or better.  tdes.Key = pdb.CryptDeriveKey("TripleDES", "SHA1", 192, tdes.IV);  Console.WriteLine("Operation complete.");  var text = Encoding.UTF8.GetString(tdes.Key);  File.WriteAllText("C:\\A folder for your deployment\\Secret.txt", text);  }  catch (Exception e)  {  Console.WriteLine(e.Message);  }  finally  {  // Clear the buffers  ClearBytes(pwd);  ClearBytes(salt);  // Clear the key.  tdes.Clear();  }  Console.ReadLine();  }  //////////////////////////////////////////////////////////  // Helper methods:  // CreateRandomSalt: Generates a random salt value of the  // specified length.  //  // ClearBytes: Clear the bytes in a buffer so they can't  // later be read from memory.  //////////////////////////////////////////////////////////  public static byte[] CreateRandomSalt(int length)  {  // Create a buffer  byte[] randBytes;  if (length >= 1)  {  randBytes = new byte[length];  }  else  {  randBytes = new byte[1];  }  // Create a new RNGCryptoServiceProvider.  var rand = new RNGCryptoServiceProvider();  // Fill the buffer with random bytes.  rand.GetBytes(randBytes);  // return the bytes.  return randBytes;  }  public static void ClearBytes(byte[] buffer)  {  // Check arguments.  if (buffer == null)  {  throw new ArgumentException("buffer");  }  // Set each byte in the buffer to 0.  for (var x = 0; x < buffer.Length; x++)  {  buffer[x] = 0;  }  }  }  } |

The “Program.cs” file should look as so, to ensure that it copied correctly check that the status, located at the bottom left hand side of the file screen should display “No issues found”:

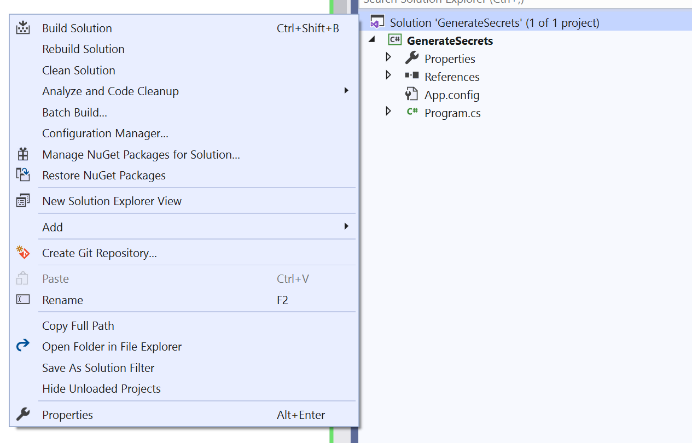


The console application saves the required value to a .txt file, the name of that txt file and its location is set under the line that starts with “File.WriteAllText(“. To change the folder location, change these values accordingly:

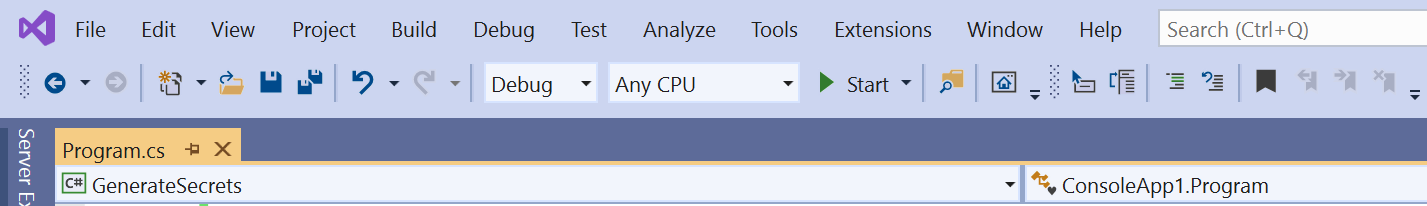


|  |
| --- |
| Note: Typical folder security will prevent you from using the root C:\ drive as the destination, instead use a sub folder similar to the example shown above |

Save and then build your project, from Solution explorer right click on the solution and select “Build Solution”. No errors should be presented:



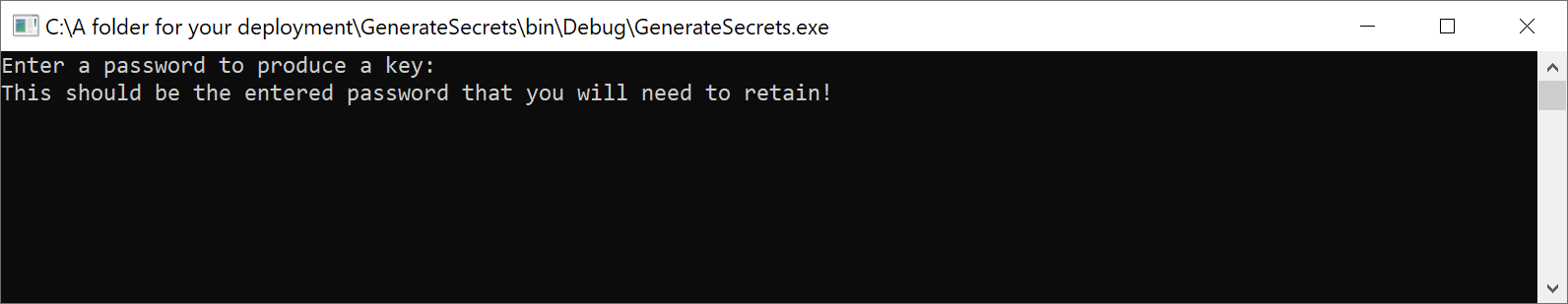
Start the application to feed the unencrypted value. Simply select “Start”:



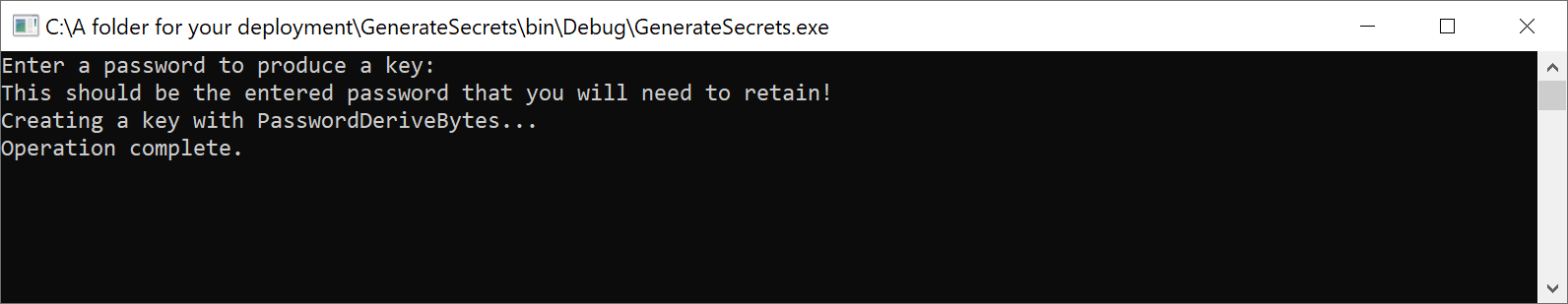
When the console appears, enter in the unencrypted key, this key will be used in the configuration record in Fundraising and Engagement to connect to the API in Azure. In this example the key to be used will be:

“*This should be the entered password that you will need to retain!*”

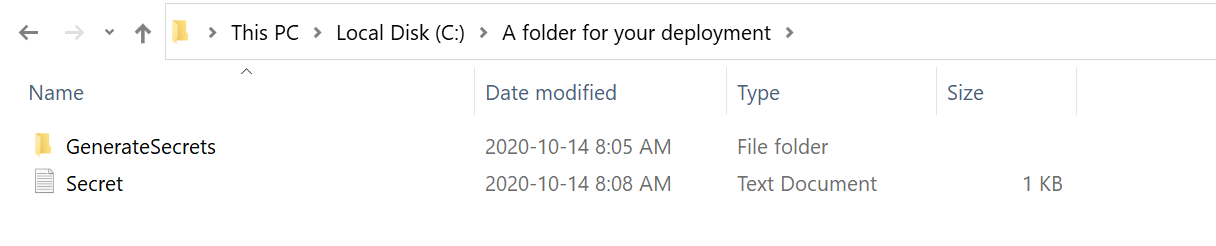
The console would then appear as:



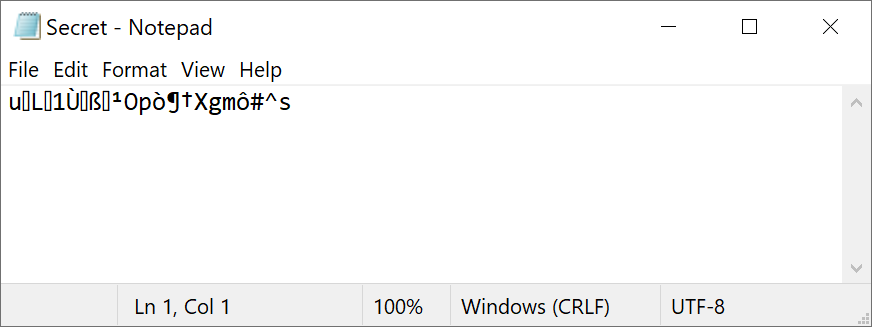
With the unencrypted key entered, press enter to generate the encrypted counterpart:



Validate the file appears in the location set in the txt file and folder destination set in the console:



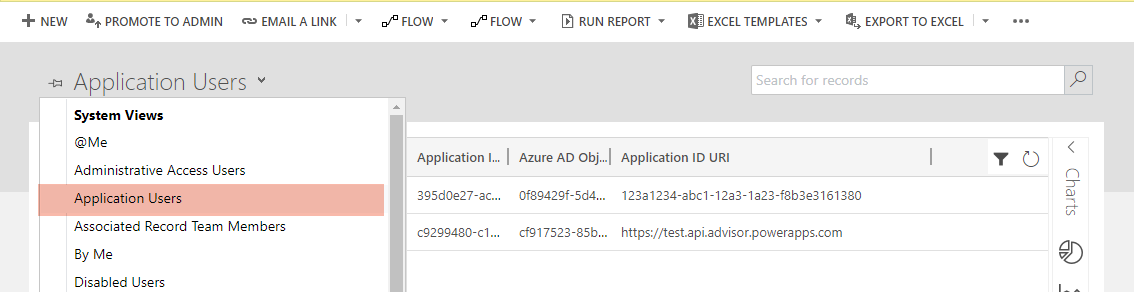
Validate the contents of the txt file by opening the file:



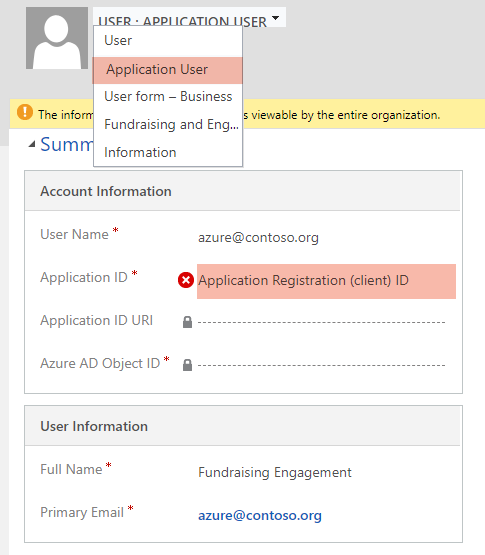
Ensure you retain the original unencrypted passkey, this will be required to complete your Fundraising and Engagement setup of the API.

## Configuring Your Application Registration in Dynamics

To ensure your Azure application registration works as expected, add the application registration into Dynamics 365. Dynamics 365 treats application registrations differently than that of regular provisioned users. To set up access, open settings for your Dynamics environment, go to Advanced Settings -> Security -> Users. Once there, change from the standard ‘All Users’ filter to ‘Application Users’:



From here you can ‘add’ the application registration by selecting the ‘Application User’ view and entering the ‘Application (client) ID’ value from [prerequisites](#_Azure_Active_Directory) to ‘Application ID’ field. Although the user name, full name, and email address are required, the application does not validate those fields so a user can enter any identifying information they please. Save the result with the save icon in the bottom right corner.



For additional details on configuring your ‘Application Users’ please review documentation on [setting up non interactive users as application users in Dynamics 365](https://docs.microsoft.com/en-us/power-platform/admin/create-users-assign-online-security-roles#create-an-application-user).

When an ‘Application User’ interacts with Dynamics 365, the application applies the same privilege and access logic as it does to an interactive user. This means an administrator must grant the application user the correct privileges for the records that will be created or updated. In Fundraising and Engagement, the following records are both created and/or updated:

* Transaction
* Payment Schedule
* Payment Method
* Response Record.

For more details on administering privileges in Dynamics 365, view [Security Roles and Privileges](https://docs.microsoft.com/en-us/power-platform/admin/security-roles-privileges).

# Deploying Fundraising and Engagement for Dynamics 365 Sales Enterprise

The intended deployment methodology is through AppSource. As the AppSource deployment is automated, it is important to note the prerequisite managed solutions which are installed or upgraded during the AppSource deployment.

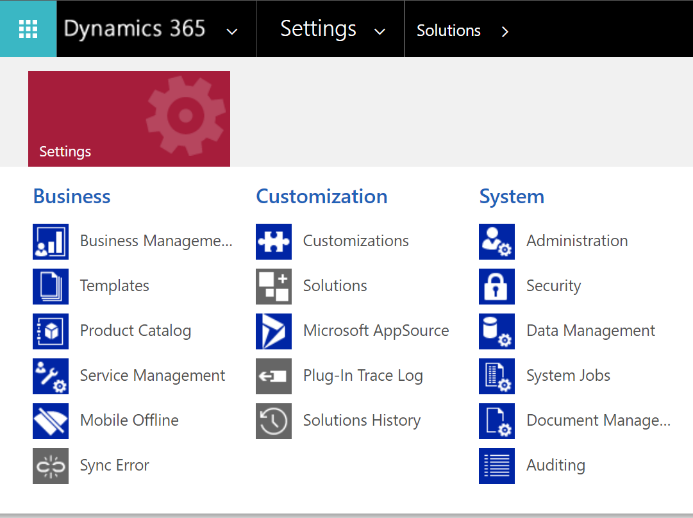
## Prerequisites Installed During Solution Deployment

As part of the required prerequisites, the Nonprofit Core managed solution is deployed. Fundraising and Engagement is set up to work with the latest version of the Nonprofit Core data schema in the Common Data Model (CDM) for Nonprofits. The Nonprofit Core can be installed directly from the Fundraising and Engagement AppSource listing.

|  |
| --- |
| Note: To inspect the latest version of the Common Data Model (CDM) for Nonprofits and how it may impact those organizations who have already deployed the CDM, please review the latest release on [GitHub](https://github.com/microsoft/Industry-Accelerator-Nonprofit/releases). |

## Acquire Fundraising and Engagement from within Dynamics

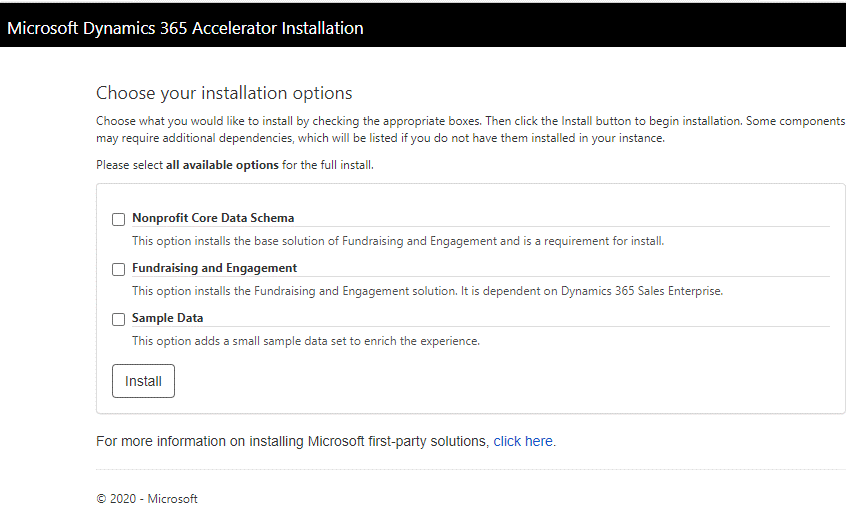
1. Login into your Dynamics 365 instance. From the application welcome screen. Select ‘Advanced Settings’, ‘Customization’ and then ‘AppSource’:



1. Using the navigation, find ‘Fundraising and Engagement’ and then select ‘Get it Now.’



1. Follow the prompted guide to complete the process. Ensure that the user completing the AppSource process has at least ‘System Administrator’ role assigned.



The Nonprofit Core Data Schema is required for Fundraising and Engagement. The Sample Data is optional and will provide a small set of sample data as well as default Configuration Record for users to use to get a good sense of how the solution works in their own environment. The Sample Data cannot be downloaded on its own. It must be deployed at the same time as the Fundraising and Engagement solution.

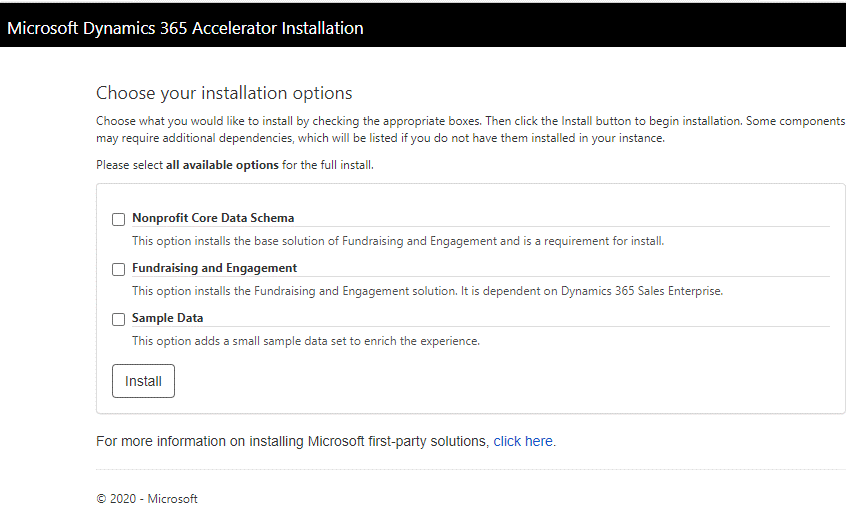
1. The Fundraising and Engagement application will appear once installed. With the application visible, this indicates a successful deployment.

## Acquire from AppSource

1. From the AppSource catalogue, navigate to Fundraising and Engagement by first going to the AppSource home page. Then search for ‘Fundraising and Engagement.’ Once located, select ‘GET IT NOW’.



1. Follow the prompted guide to complete the process ensuring that the user completing the AppSource process has at least ‘System Administrator’ role assigned.
2. You will be prompted to login to your tenant. From there select the Dynamics 365 instance you want to deploy Fundraising and Engagement. Follow the remaining prompted steps, such as the privacy policy and terms and conditions, both of which must be accepted for the solution to be installed.



The Nonprofit Core Data Schema is required for Fundraising and Engagement. The Sample Data is optional and will provide a small set of sample data as well as the default Configuration Record for users to get a good sense of how the solution works in their own environment. The Sample Data cannot be downloaded on its own. It must be deployed at the same time as the Fundraising and Engagement solution.

1. The Fundraising and Engagement application will appear once installed. With the application visible, this indicates a successful deployment.

Once Fundraising and Engagement is successfully deployed, the user needs to set up the Configuration Record for the Dynamics components to work properly. Step by step instructions on how to configure the Configuration Record can be found in the [User Guide](https://aka.ms/feuserguide).

# Provisioning the ARM Template

In this step, you will create the Azure components required by Fundraising and Engagement using a supplied [ARM template](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/overview).

## Azure Components Created

The following Azure components are created during the deployment of the ARM template.

|  |  |  |
| --- | --- | --- |
| Component | Type | Purpose |
| API | App Service | The API used to send data from Dynamics 365 Customer Engagement to Azure. |
| Background Services | App Service | The Function processes to manage rollup queries and calculations, this primarily writes values back to Dynamics 365 Customer Engagement. |
| Bank Run | App Service | The application which generates and delivers the import file for the specified bank using the specified value. |
| Recurring Donation Engine | App Service | A process that manages the recurring gift process and generates the results in the form of Transactions that are written back to Dynamics 365 Customer Engagement. |
| App Service Plan | App Service Plan | The service plan created during the creation of the Azure environment. This plan can be edited after creation. |
| API Insights | Application Insights | The corresponding Application Insights instance that corresponds to the created API. |
| Background Services Insights | Application Insights | The corresponding Application Insights instance that corresponds to the created Background Services. |
| Bank Run Insights | Application Insights | The corresponding Application Insights instance which corresponds to the created Bank Run. |
| Recurring Donation Engine Insights | Application Insights | The corresponding Application Insights instance which corresponds to the created Recurring Donation Engine. |
| Vault | Key Vault | The secure storage that brokers sensitive information such as SQL passwords between Azure Applications. |
| Azure SQL Database | SQL Database | The Azure SQL Database which stores data passed to it from Dynamics. |
| Azure SQL Server | SQL Server | The Azure SQL server that is created by default in order to facilitate the Azure SQL Database. |
| Logging Storage | Storage Account | The storage account which holds the logging information from the recurring donation engine. |

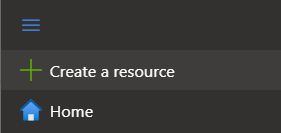
## Setting up Azure Components to Support Multiple Dynamics Environments

The deployment template creates a single resource group per instance of Dynamics 365. An administrator preparing an implementation spanning several Dynamics 365 instances may wish to share a single resource group. In this case, the template may act as the starting process for the deployment and administrators can expand this by manually adding additional resources.

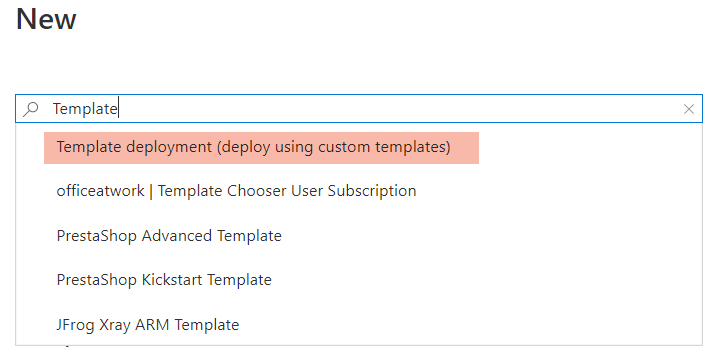
Note that this template and this guide cover a typical deployment for a single instance of Dynamics 365.

## Deploying the ARM Template

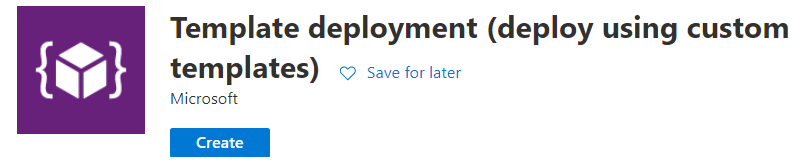
1. Login into your Azure Portal as the user with ‘Owner’ privileges for the subscription. The guide assumes you have accessed your Azure environment via <https://portal.azure.com>.
2. Create a new resource.



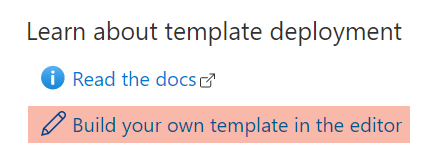
1. In the search box, enter ‘Template’ and select ‘Template deployment (deploy using custom templates).’



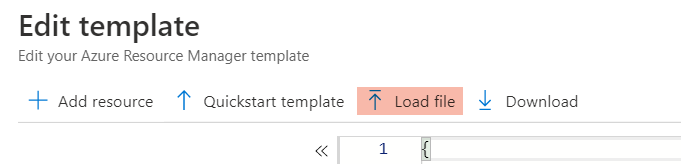
1. Select, ‘Create.’



1. Select ‘Build your own template.’



1. Select ‘Load file’ from within the ‘Edit template’ section and select the ARM Template (available [here](https://aka.ms/fe-armtemplate)).



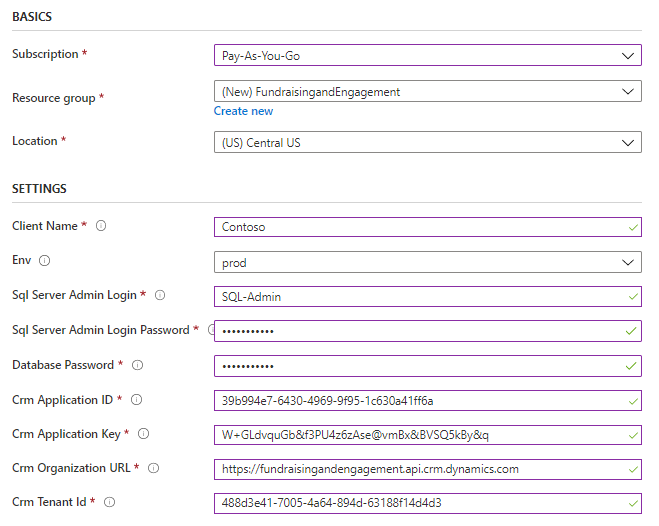
1. Select ‘Save’ to progress to the user inputs.
2. The template will require all inputs to be completed before the deployment. The specific inputs are:
   1. Basics section

|  |  |  |
| --- | --- | --- |
| **Setting** | **Description** | **Example** |
| *Subscription* | Select from an existing subscription, although multiple subscriptions may be visible, only those that the user deploying the template as the ‘Owner’ role set on can be used. | *Pay-As-You-Go* |
| *Resource Group* | Enter the resource group to be used, enter a descriptive name as all the Azure components will be assigned to this resource group and identifying those resources that are needed for Fundraising and Engagement will be beneficial to a demonstrator.  The resource group is assumed to already exist before deployment. | *FundraisingandEngagement* |
| *Location* | The resource locations for these components. All resources are assigned to specific geographic locations based on the group. Locations can be moved post deployment. | *(US) Central US* |

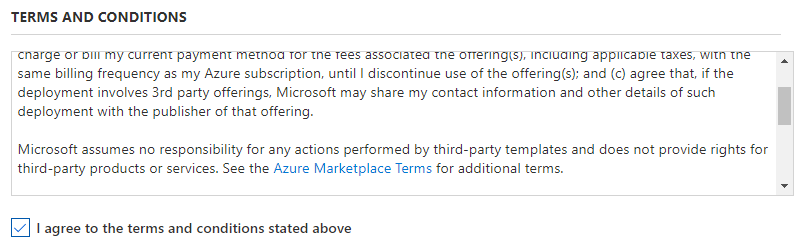
* 1. Settings section

|  |  |  |
| --- | --- | --- |
| **Setting** | **Description** | **Example** |
| *Client Name* | The short name for the client. This value acts as the suffix prefix applied to the components created during this process forming a unique name for each application.  Because the value is used as the prefix of resource names, some limitations apply. E.g., the value can contain only numbers and letters and the maximum length is limited. | *Contoso* |
| *Env* | Select from Dev, QA, UAT, Test and Prod. Select the option that best suits the purpose of this deployment. This acts as a prefix to the naming of the created applications. | *prod* |
| *Sql Server Admin Login* | The SQL Server Administrator account that will be used when creating the Azure SQL Server. It will be important to record these details securely as they are required to complete additional configuration. | *sql-admin* |
| *Sql Server Admin Login Password* | The SQL Server Administrator password that will be used when creating the Azure SQL Server. It will be important to record these details securely as they are required to complete additional configuration. | *An example password* |
| *Database Password* | The SQL Database user password that will be stored as a secret in the secure key vault. | *An example password* |
| *CrmApplication ID* | This is the ‘Application (Client)’ ID captured during the [prerequisites](#_Azure_Active_Directory). | *39b994e7-6430-4969-9…* |
| *Crm Application Key* | The application secret key obtained from the ‘Application’ registration in Azure Active Directory. Captured during your [prerequisite](#_Azure_Active_Directory) checklist. | *W+GLdvquGb&f3PU4z6…* |
| *Crm Organization URL* | The Dynamics 365 for Sales Enterprise API URL. | *https://fundraisingandengagement.api.crm.dynamics.com* |
| *Crm Tenant Id* | The Azure Active Directory identifier. | *488d3e41-7005-4a64-8…* |

1. Using the above example inputs, the template will look like this:



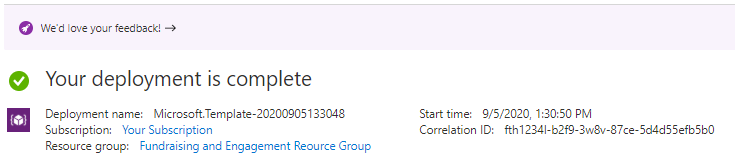
1. To start the process, select ‘I agree to the terms and conditions stated above.’



1. Proceed to create.
2. Review the completed process. Any errors presented during the deployment of the template will result in an administrator needing to delete the components, correct the input error, and re-run the template.
3. The process will create several resources, including the following:

|  |  |
| --- | --- |
| **Azure Resource** | **Example Name (Using the Above Example)** |
| Service Plan | Contoso-AppServicePlan |
| SQL Server | Contoso-sqlserver-prod |
| SQL Database | Contoso-db-prod |
| SQL Database Username | Contoso\_prod\_user |
| Secure Key Vault | Contoso-vault-prod |
| Azure Storage Account | Contosostorageprod |
| Azure Function | Contoso-BackgroundServices-prod |
| Azure Web Application (API) | Contoso-ApiApp-prod |
| Azure Web Job (Recurring Donation Engine) | Contoso-RecurringDonationApp-prod |

1. The deployment will alert the user to the success of the deployment. A successful deployment presents the following:



## Adding the Azure SQL User Accounts

Once the template has successfully completed the creation of all Azure components, the pre-created SQL user account needs to be set up within the Azure SQL Database.

|  |
| --- |
| Note: SQL Server Management Studio is required as these steps must be performed against the ‘master’ database of the SQL Azure Server. You can download the latest version of [SQL Server Management studio](https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15). To enable access from SQL Server Management Studio to the SQL Azure Server, follow these steps on how to set firewall rules using the [Azure portal](https://docs.microsoft.com/en-us/azure/azure-sql/database/firewall-configure#use-the-azure-portal-to-manage-server-level-ip-firewall-rules). |

The SQL username will have been created using a pre-defined naming convention ‘<Client Name>\_<Env>\_User’ where <Client Name> and <Env> to the respective parameters of the ARM template. Using the examples listed above, the SQL user account will be:

*Contoso\_prod\_user*

Run this script against **master** database:

CREATE LOGIN SQL\_Login\_Name (e.g. Contoso\_prod\_user)

WITH PASSWORD = 'Password as entered during ARM Template configuration'

GO

With the SQL user setup within the SQL Server, the SQL user must now be provisioned to access the SQL Database. Run this script against the **provisioned** database:

CREATE USER SQL\_Login\_Name e.g. Contoso\_prod\_user

FOR LOGIN SQL\_Login\_Name e.g. Contoso\_prod\_user

WITH DEFAULT\_SCHEMA=dbo

GO

EXEC sp\_addrolemember N'db\_owner', N'SQL\_Login\_Name e.g. Contoso\_prod\_user'

GO

# Deploying the Fundraising and Engagement components to Azure

These steps assume the Azure environment is now ready to be populated by the applications that power the Fundraising and Engagement solution. Each component created must be in place before completing the steps outlined in this section.

The following prerequisites are required to deploy the template and create the Azure components:

* Visual Studio 2019 Community Edition or Higher
* Access to the latest [GitHub repository for Fundraising and Engagement](https://github.com/microsoft/fundraising-and-engagement)
* The latest updates to [Visual Studio 2019 Community Edition](https://visualstudio.microsoft.com/vs/community/) or higher.

|  |
| --- |
| Note: This guide assumes the user populating the Azure environment has a basic understanding of Visual Studio. Visual Studio 2019 Community Edition or higher is the only supported version. Download the latest version of [Visual Studio 2019](https://visualstudio.microsoft.com/downloads/), a free development tool. |

## Clone the Solution

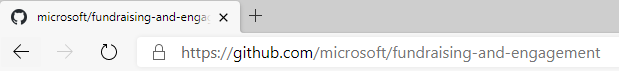
The goal is to clone the latest GitHub repository to your local Visual Studio instance. This guide covers only the initial clone.

|  |
| --- |
| Note: If you have already deployed the Fundraising and Engagement Azure components using this guide, connect to [GitHub](https://github.com/microsoft/Industry-Accelerator-Nonprofit/releases) to retrieve the latest solution. Always use the most up to date version of the solution when deploying to Azure. |

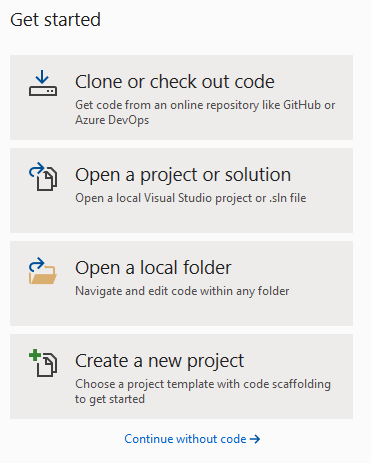
1. Retrieve the latest GitHub repository URL by locating the most recent repository and selecting ‘<> Code.’



Copy the URL that now displays in the browser window.

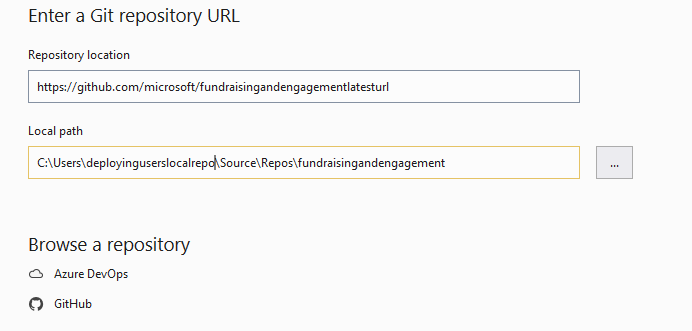


1. Open Visual Studio and select ‘Clone or check out code.’

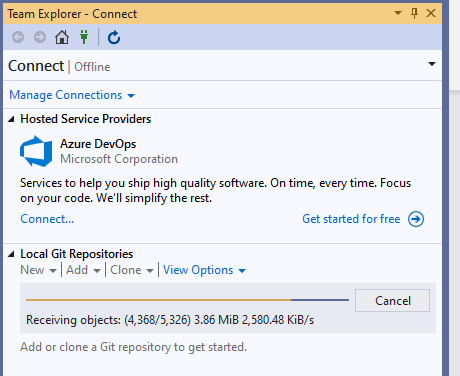


1. Enter the previously copied GitHub URL. You now have an option to change the local path.

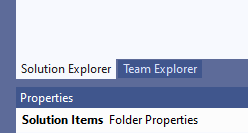
|  |
| --- |
| Note: If you have already deployed the Fundraising and Engagement Azure components using this guide, to avoid a conflict, you may want to change the local path to indicate the most recently cloned release from GitHub. |



1. The solution will indicate it is cloning successfully by presenting the progress bar.



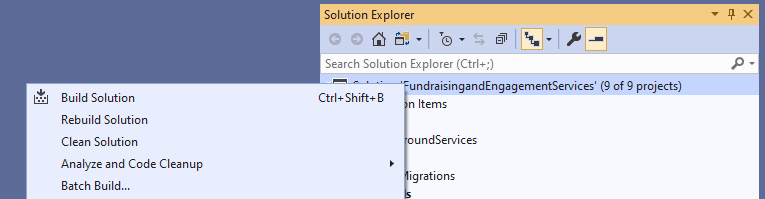
1. Change the view to default on the ‘Team Explorer’ tab.
2. Open the services solution by double clicking on the ‘FundraisingandEngagementServices.sln.’
3. Navigate back to the solution by selecting the ‘Solution Explorer’ tab.



## Build the Solution

The solution needs to be built locally prior to deploying to Azure. The process of building the solution to deploy does not require running it locally (only the build step is required).

To build the solution, right click on the solution and select ‘Build Solution.’



There are nine build steps. The user can confirm that the build was successful by verifying the output:

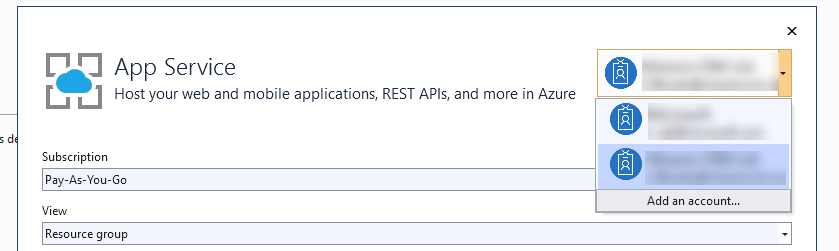
Sample screenshot of step #9 showing successful verification

## Deploy to Azure

Deploying the solution to the previously created Azure components requires that the user deploying the solution has access to those that were previously created. This guide assumes the user creating the Azure components is using the same access credentials as the user that then deploys the Fundraising and Engagement solution.

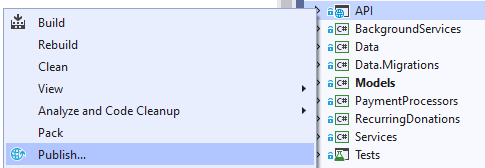
There are three components that require deploying to Azure: the API, RecurringDonations, and the BackgroundServices. Follow these steps to deploy each Azure application.

Note: Visual Studio 2019 Community Edition supports having multiple profiles for Azure deployment. If you already have some profiles, make sure the profile in use matches the desired target for deployment:

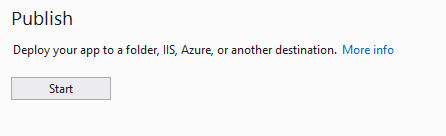


### API

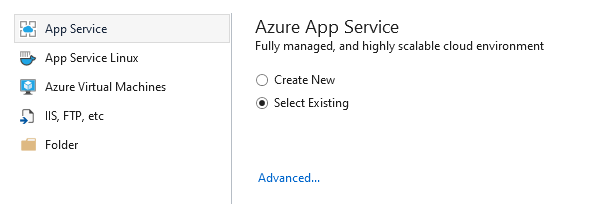
1. Right click on the API Visual Studio folder and select ‘Publish.’



Select ‘Start.’



1. Select ‘App Service’ (this is the default) and ensure that ‘Select Existing’ is checked.



1. Select ‘Create’ and ensure the profile in use is the profile that has ‘contributor’ rights to the Azure deployment. Select the subscriptions, change the view to ‘Resource Group,’ then locate the ‘API’ created during the Azure deployment steps.



1. Select ‘OK’ then ‘Publish.’

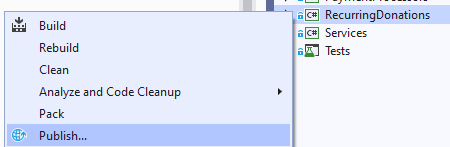


1. Verify the success by ensuring the ‘Output’ section displays the successful deployment.

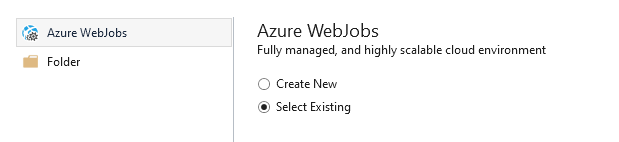


### Recurring Donation

1. Right click on the ‘RecurringDonations’ Visual Studio folder and select ‘Publish.’



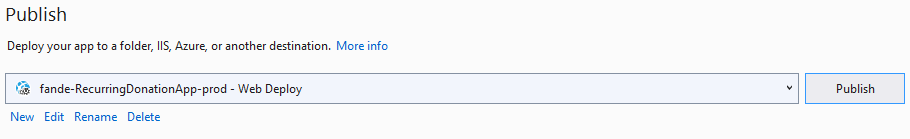
1. Select ‘Azure WebJobs’ then ‘Select Existing.’



1. Select ‘Finish’ and ensure the profile in use is the profile that has ‘contributor’ rights to the Azure deployment. Select the subscriptions, change the view to ‘Resource Group,’ then locate the ‘Recurring Donation App’ created during the Azure deployment steps.



1. Select ‘OK’ then ‘Publish.’



1. Verify success by ensuring the ‘Output’ section displays the successful deployment.

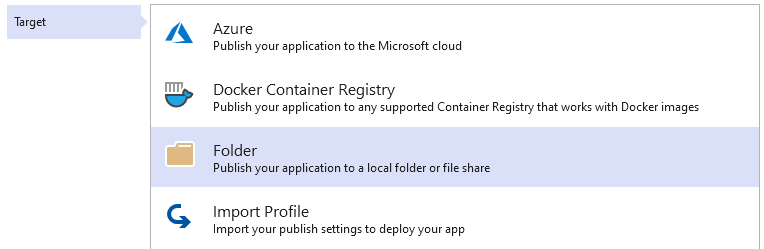


### Background Services

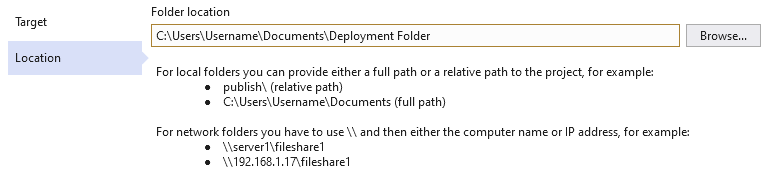
1. Right click on the ‘BackgroundServices’ folder in Visual Studio folder and select ‘Publish.’



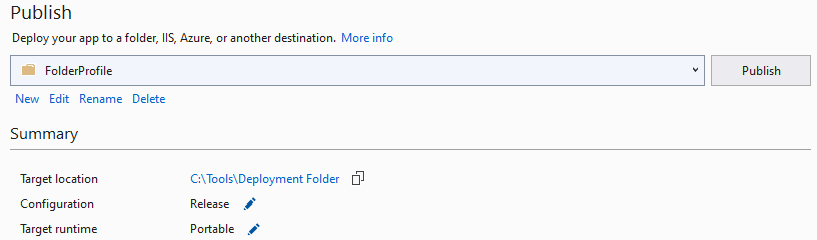
1. Select ‘Folder’ then ‘Next.’



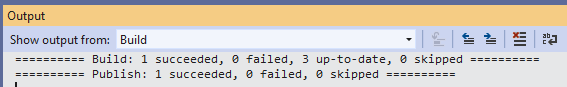
1. Select a local folder repository that you have permission to write to e.g. C:\Users\Username\Documents\Deployment Folder.



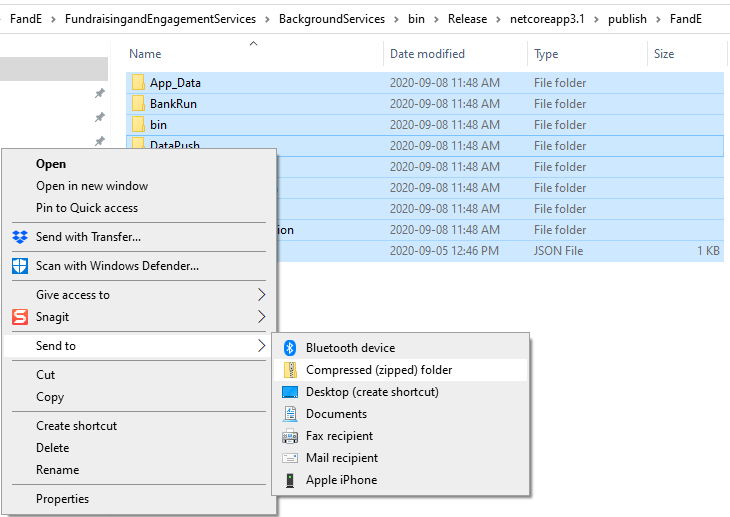
1. Select ‘Publish.’



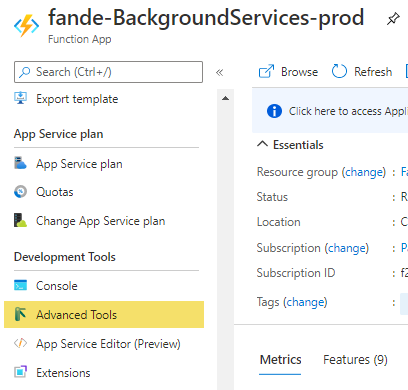
1. Confirm the build was successful.



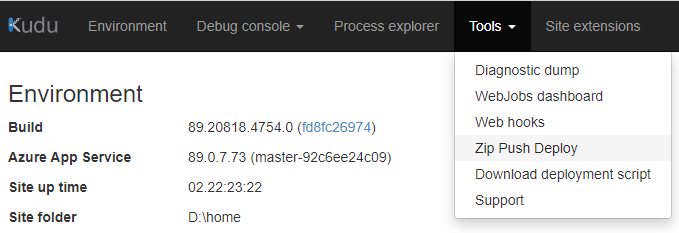
1. Navigate to the deployment folder, select all folders in the deployment path, and add them to a zip file.



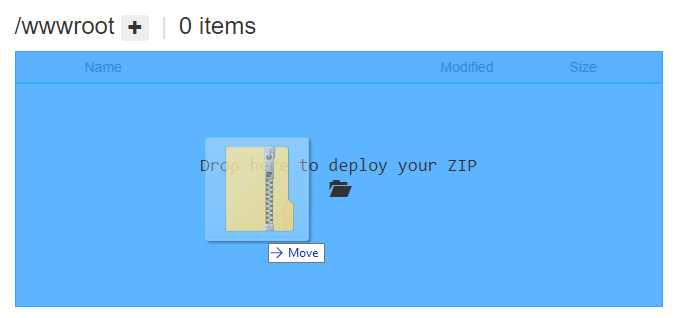
1. Navigate to the originally created function app within Azure, select ‘Advanced Tools,’ then select ‘Go.’



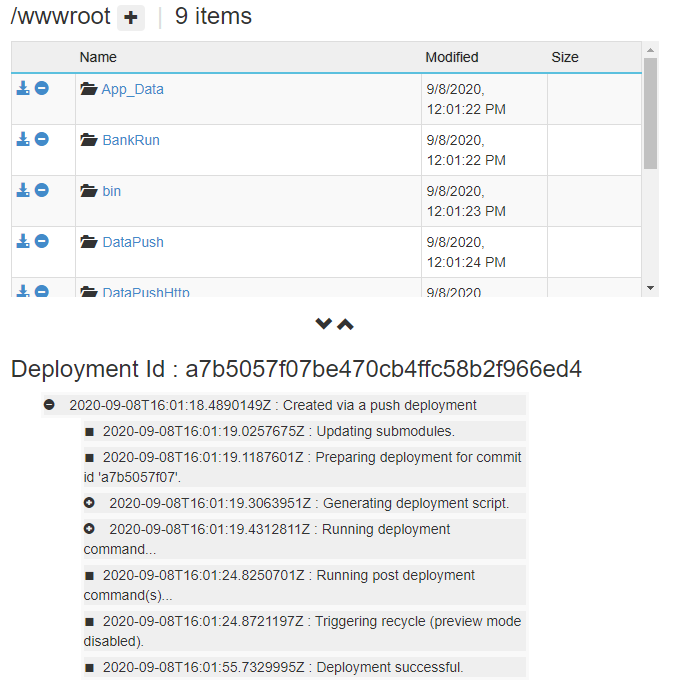
1. From the explorer tools, select ‘Tools’ then ‘Zip Push Deploy.’



1. Drag and drop the zip file directly to the open browser window.



1. The process will complete and display a successful deployment message.



1. Your deployment is complete. All three Azure components are now deployed.

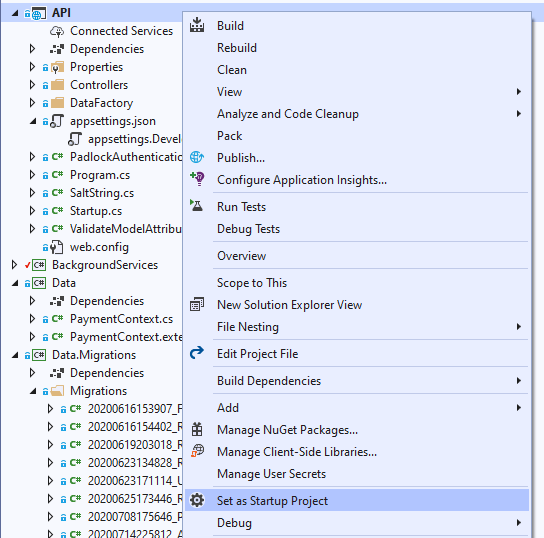
## Populate the Azure SQL Schema (Using C# Migrations)

To populate the newly created database using the C# migrations from within Visual Studio, follow these steps.

|  |
| --- |
| Note: This assumes you have access to the details of the created Azure components and have successfully opened the solution in Visual Studio. |

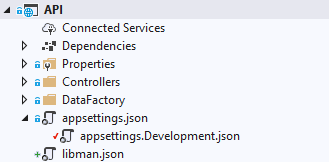
* + - 1. **Set the API as the Startup Project**

From within the project navigation, highlight the “API” project and right click, then select ‘Make Start Up Project.’



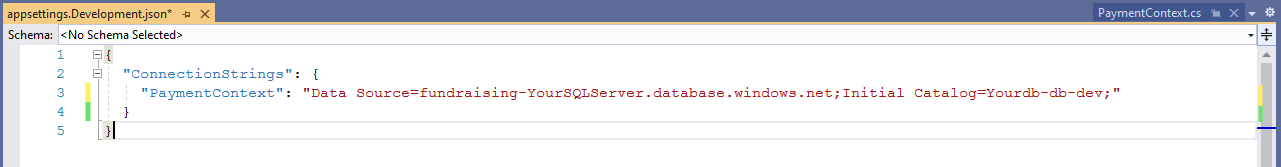
* + - 1. **Update the appsettings.Development.json file**

Locate the appsettings.Development.json file in the API project. Double click on this file to review its contents.



* + - 1. **Set the PaymentContext data source**

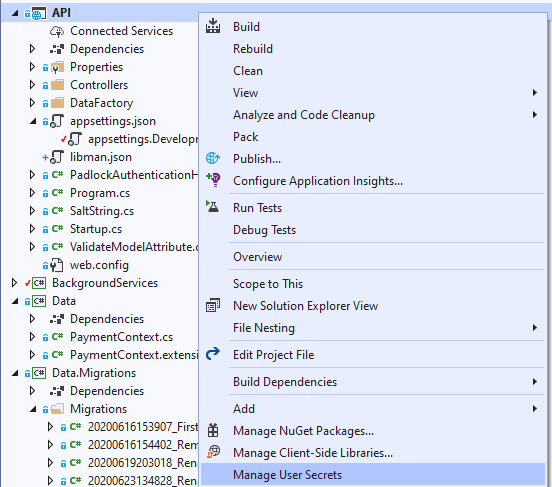
With the appsettings.Development.json open, update the connection string with the details captured during the Azure setup.



The PaymentContext value should consist of the SQL Server with the intial catalogue as the name of the database created during the process. In the above example this would be:

* Data Source: Contoso-sqlserver-prod
* Initial Catalogue: Contoso-db-prod.
  + - 1. **Create the secrets.json**

To create or manage the private secrets needed to send the password information to build the SQL tables, right click on the ‘API’ project and then select “Manage User Secrets”. Note that if no user secrets have been stored prior, this may read ‘Add User Secrets’.



With the user secrets open, assuming no existing user secrets already exist, copy and paste the following into it:

{

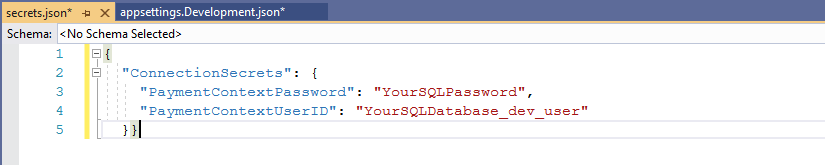
"ConnectionSecrets": {

"PaymentContextPassword": "YourSQLPassword",

"PaymentContextUserID": "YourSQLDatabase\_dev\_user"

}}

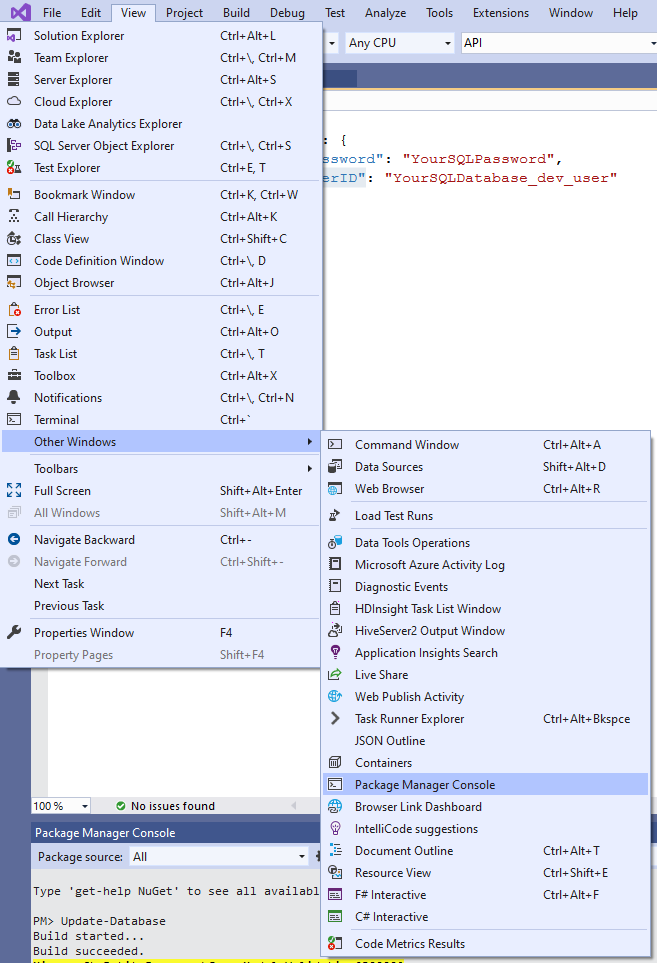
The following result should appear:



The two values required that are specific to each environment will be:

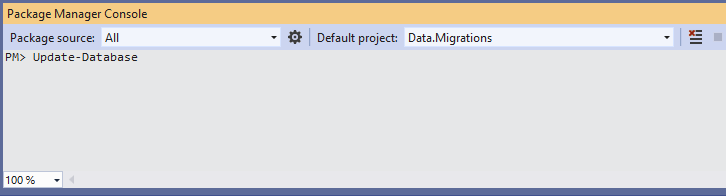
1. "PaymentContextPassword": The SQL user password recorded during the setup of the Azure components.
2. "PaymentContextUserID": The SQL username recorded during the setup of the Azure components.
   * + 1. **Navigate to Package Manager Console**

With the connections now updated, the SQL tables can be created using the C# migrations within the solution. Navigate to ‘View’, then ‘Other Windows’ and then select ‘Package Manger.’

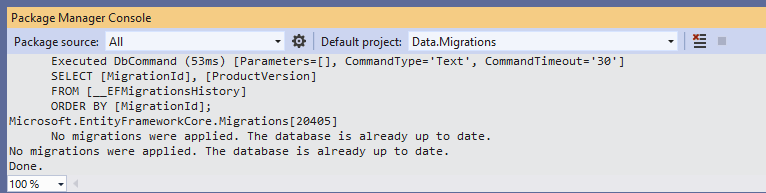


* + - 1. **Run Migrations**

Within package manager, type in ‘Update-Database.’



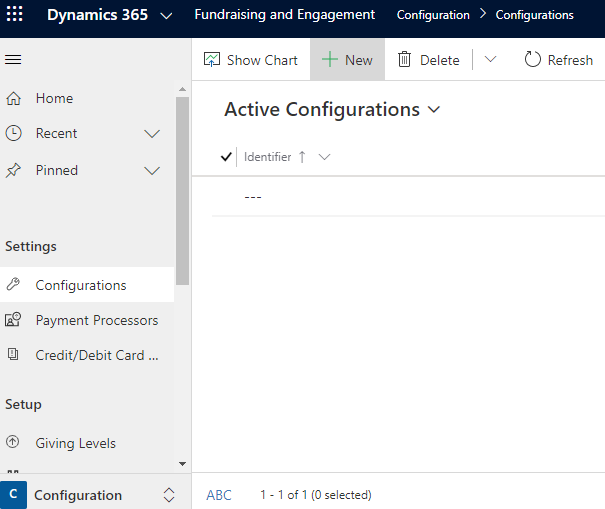
Hit Enter, and then confirm that the last message presented from the Package Manager reads ‘Done’. This indicates a successful deployment of the tables and schema changes.



# Post Deployment Tasks (Dynamics 365)

Additional key steps are required after an administrator has set up the Fundraising and Engagement Dynamics 365 environment. At least one configuration record populated with needed Azure information is required and all system users must have a configuration record assigned to their user record prior to use.

The configuration record can be created and edited in the Fundraising and Engagement application. Go to the ‘Configuration’ area (at the bottom left of the navigation pane) and select ‘Configurations’



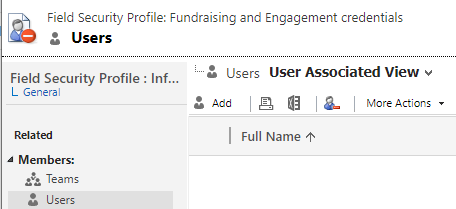
## Assigning Security Roles

Some configuration is necessary before all features of Fundraising and Engagement can be used, the Configuration record in particular needs to be created. In case this is not done by a user with the System Administrator role, the user needs both the appropriate role and field security profile:

* The ‘FundraisingandEngagement: System Administrator’ role



* The ‘Fundraising and Engagement credentials’ field security profile (Managed in Advanced settings > Security > Field Security Profiles)



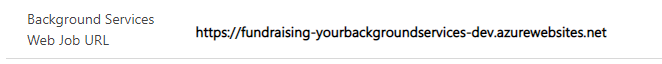
## Configuration Record Prerequisites

The Configuration Record requires several specific Azure-dependent configuration settings to be set prior to use. These five configuration settings have been listed here along with how to retrieve the value from Azure. Note that the labels of the following options match the label of the configuration record in Fundraising and Engagement.

1. **Azure Web API URL**



1. Locate the Azure Web App that is hosting your API components.
2. From the General tab, locate the URL.
3. Paste the URL into the ‘Azure Web API URL’ field and append /api/ to it. The full URL must read: <https://fundraising-yourexample-dev.azurewebsites.net/api/>.
4. **API Padlock Token**
5. Currently, the padlock token is set to a default value which must be changed. Enter the unencrypted padlock token (passcode) generated in the [prerequisites above](#_Generating_a_Padlock).
6. If you update the Padlock Token value after the initial deployment, you need to put the salted Padlock Token value in the APIKeys:SaltedGatewayAPIKey configuration entry in the Azure API application.
7. **Background Services Web Job URL**



1. Locate the Azure Background Functions application.
2. Under Overview, view the URL.
3. Paste the URL into the ‘Background Services Web Job URL’ field as is – the full URL must read: <https://fundraising-yourbackgroundservices-dev.azurewebsites.net>.
4. **Bank Run Security Key**



1. Locate the Azure Background Functions application.
2. Under Functions, select Functions.
3. Navigate to and select the ‘BankRun’ function.
4. Once the function opens in the new window, select ‘Overview.’
5. Select ‘Get Function Url.’
6. In the URL, copy the entire value after the ‘?code=’
7. Update the ‘Bank Run Security Key’ value with that of the copied URL value. The result should read:gmai *a5RSdPgNdTKyUMEZe2BCHsdkcmonm4t8ngfnwmc2o5Pu99g78ks7RzuiHnmODbECfPSQ==*
8. **Yearly Giving Security Key**



1. Locate the Azure Background Functions application.
2. Under Functions, select Functions.
3. Navigate to and select the ‘YearlyGivingFromEntity’ function.
4. Once the function opens in the new window, select ‘Overview.’
5. Select ‘Get Function Url.’
6. In the URL, copy the entire value after the ‘?code=’
7. Update the ‘Yearly Giving Security Key’ value with that of the copied URL value. The result should read: *a5RSdPgNdTKyUMEZe2BCHsdkcmonm4t8ngfnwmc2o5Pu99g78ks7RzuiHnmODbECfPSQ==*
8. **Event Receipting Security Key**

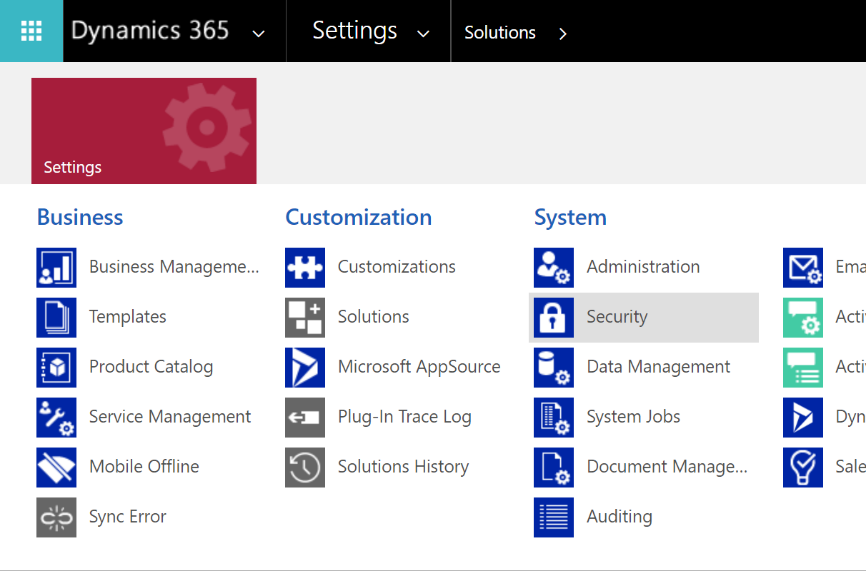


1. Locate the Azure Background Functions application.
2. Under Functions, select Functions.
3. Navigate to and select the ‘EventReceipting’ function.
4. Once the function opens in the new window, select ‘Overview.’
5. Select ‘Get Function Url.’
6. In the URL, copy the entire value after the ‘?code=’
7. Update the ‘Event Receipting Security Key’ value with that of the copied URL value. The result should read: *a5RSdPgNdTKyUMEZe2BCHsdkcmonm4t8ngfnwmc2o5Pu99g78ks7RzuiHnmODbECfPSQ==*

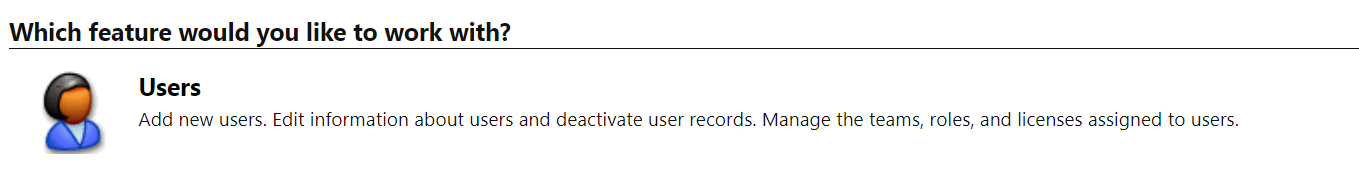
## Assigning Configuration Settings to Dynamics 365 Users

All users that need to interact with the Fundraising and Engagement business logic must have a configuration record assigned, **including the application user** [configured above](#_Configuring_Your_Application). Users are limited to a single configuration record assigned at any one time. To associate a configuration record to a user in Dynamics, log in to Dynamics 365 as a user that has the System Administrator role and complete the following steps.

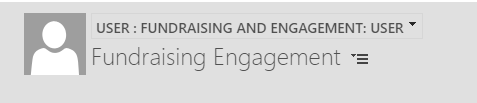
1. Navigate to ‘Advanced Settings.’
2. Navigate to System and then select ‘Security.’



1. Select Users.



1. Retrieve and open the user record you want to assign the configuration record to.
2. Select the ‘Fundraising and Engagement: User’ view



1. Use the lookup to set the correct configuration record in the ‘Configuration’ field and then hit save and close.



**Sample Data Import**

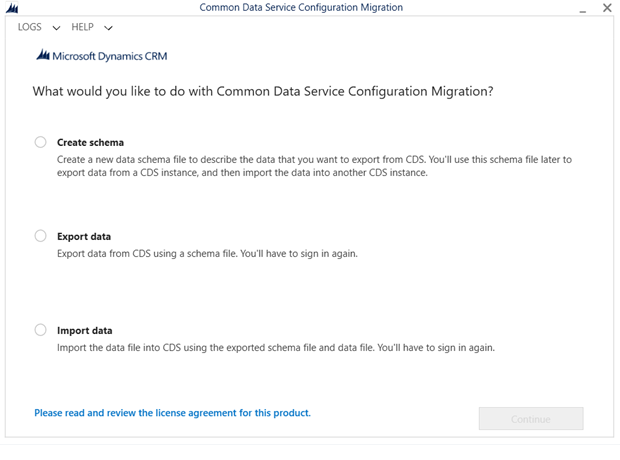
To import sample data to Fundraising and Engagement solution complete the following steps.

1. Download a sample dataset from the Fundraising and Engagement GitHub repository. There is *FundraisingAndEngagement.Sample.Data.zip* file in the S*amples* folder.
2. Make sure a correctly configured configuration record exists in the application prior to data import (Web API URL, Padlock token are required, as well as the association with the user account).
3. Go to SQL Server Management Studio to your database connection and disable foreign key constraints. You can use the following script to do that.

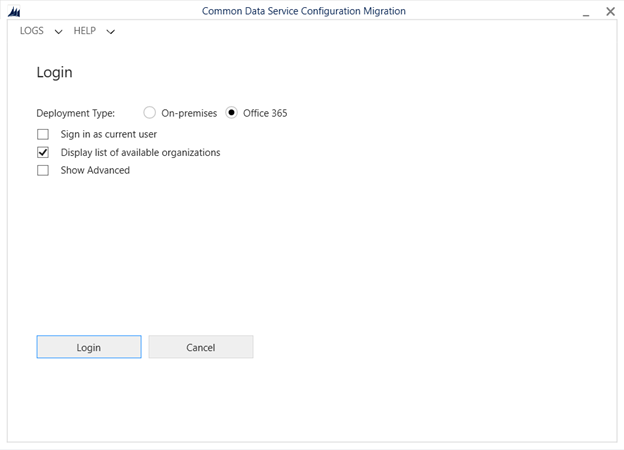
|  |
| --- |
| DECLARE @enable\_constraints bit = 0  --Don't change anything below this line.    DECLARE @schema\_name SYSNAME  DECLARE @table\_name SYSNAME    DECLARE table\_cursor CURSOR FOR  SELECT  schemas.name,  tables.name  FROM  sys.tables  INNER JOIN sys.schemas ON tables.schema\_id = schemas.schema\_id    OPEN table\_cursor  FETCH NEXT FROM table\_cursor INTO @schema\_name, @table\_name    DECLARE @cmd varchar(200)  WHILE @@FETCH\_STATUS = 0  BEGIN  SET @cmd = 'ALTER TABLE ' + QUOTENAME(@schema\_name) + '.' + QUOTENAME(@table\_name) + ' '  SET @cmd = @cmd + (CASE WHEN @enable\_constraints = 1 THEN 'CHECK' ELSE 'NOCHECK' END) + ' CONSTRAINT ALL'    PRINT @cmd  EXEC( @cmd )    FETCH NEXT FROM table\_cursor INTO @schema\_name, @table\_name  END    CLOSE table\_cursor  DEALLOCATE table\_cursor |

*Note: This script does not drop the constraints, it just disables them.*

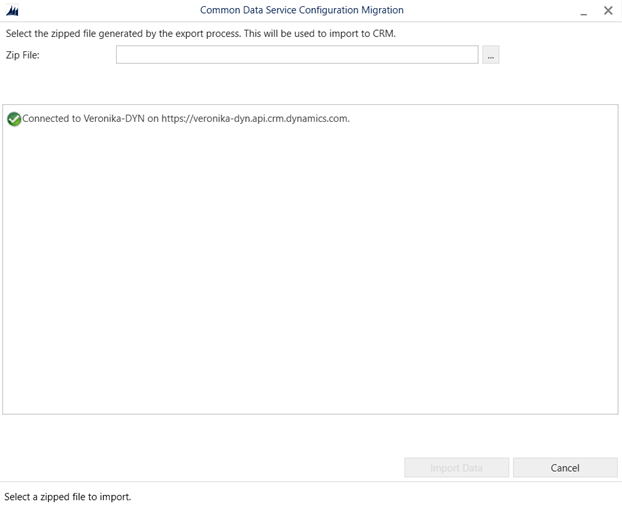
1. Continue with deploying the sample data set using the Data Migration utility. This tool is part of Power Apps developer tools. It can be downloaded here.
2. Run Data Migration utility.



1. Select the *Import Data* option.
2. Check the *Display list of available organizations* option and log in.



1. Select your environment from the list of available organizations and click Login.
2. After successful Login you will be prompt to select sample data file to import



1. Locate the downloaded FundraisingAndEngagement.Sample.Data.zip and click *Import Data.*
2. Wait until the tool finishes the import.
3. Once the tool finishes the import, check if the data is in the Azure database.
4. The last step is to re-enable foreign key constraints in the database. To re-enable the constraints, just change the first line of the script from step 3 to *DECLARE @enable\_constraints bit = 1* and run it against your database.
5. Your sample data set is set up.

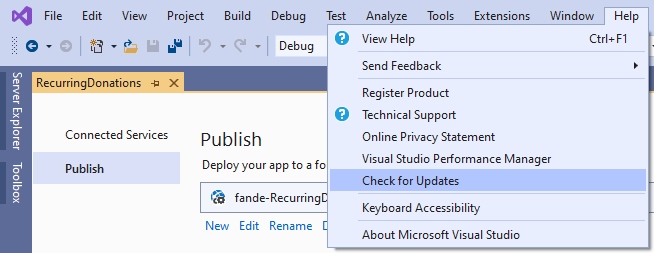
# Troubleshooting

## When a Failure Occurs

The ARM template is not capable of rolling back components. This means if the ARM template fails as a result of the prerequisites not being met, the user implementing the template must manually remove the created components prior to reattempting the deployment.

## Updating Visual Studio 2019 Community Edition (or Higher)

Ensure your installation has the latest updates. This guide relies on the integrated publishing processes from Visual Studio to Azure components. As a result, they may need to be frequently updated for Azure components to be deployed. To update Visual Studio, select ‘Help’ then ‘Check for updates.’ Follow prompts to update the installation if required.



## Modifying the Template

The ARM template has been configured and tested to deploy a single Azure environment per Dynamics 365 instance. For implementors wishing to change the deployment process or alter how the template delivers the application and configuration, they do so at their own risk. Support currently covers a single deployment per Dynamics 365 instance architecture.

## Naming and Unique Name Concepts

A failure during deployment of the template can occur due to the naming conventions used during the template setup. Resource names are externally addressable and must be unique. The template has been configured to generate unique names for the created applications but in the unlikely event that a names space for a component within the template is already in use, the deployment will fail. If this occurs, the implementor must modify the organization name to create a unique naming convention.

Please see details on best practices for [Azure naming conventions](https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/naming-and-tagging).

## Hitting Maximum Path Length Limit When Deploying Azure Components

If you are using Windows, the build or deploy of Azure components in Visual Studio may fail if path lengths of files created during the build reach the maximum Windows [path length limit](https://docs.microsoft.com/en-us/windows/win32/fileio/naming-a-file#maximum-path-length-limitation). In order to avoid this issue, please move the root of the fundraising-and-engagement git repo to a different location on filesystem with shorter path.

# Learn More About Azure and Extending Dynamics 365

This document assumes users provisioning Fundraising and Engagement Azure environments have a solid understanding of both the Azure components that will be created and configured as well as the Dynamics 365 environment which will utilize this functionality.

The following courses and certification paths are recommended prior to utilizing this guide:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Educate** | **Accredit** | **Discussion** |
| Azure Fundamentals | [Learning](https://docs.microsoft.com/en-us/learn/paths/azure-fundamentals/) Path | [Exam](https://docs.microsoft.com/en-us/learn/certifications/exams/ai-900) | [Community](https://powerusers.microsoft.com/t5/PowerApps-Community/ct-p/PowerApps1) |
| Dynamics 365 Fundamentals | [Learning](https://docs.microsoft.com/en-us/learn/paths/dynamics-365-fundamentals/) Path | [Exam](https://docs.microsoft.com/en-us/learn/certifications/exams/mb-900) | [Community](https://powerusers.microsoft.com/t5/Microsoft-Flow-Community/ct-p/FlowCommunity) |