# Azure IoT Academy Month Two, Day Two, Lab Two

## Click Below Link to Watch Lab Two Video (placeholder)

## Day Two: Hands on Lab Two Video

This hands on lab builds on the prior lab, leveraging the following to attain similar results:

- Azure Digital Twins SDK
- Visual Studio Code

Developers working with Azure Digital Twins commonly write client applications for interacting with their instance of the Azure Digital Twins service. This developer-focused lab provides an introduction to programming using the Azure Digital Twins SDK for .NET (C#) to interact with Azure Digital Twins. We will walk you through writing a C# console client app step by step, starting from scratch.

In this lab, you will...

- Create a C# project.
- Setup local Azure credentials.
- Add code to your C# project that will:
  - Create a model
  - Create digital twins
  - Create relationships
  - List relationships
  - Query digital twins

**Course Content**

# 1.1. Prerequisites

This Azure Digital Twins lab uses the command line for setup and project work. As such, you can use any code editor to walk through the exercises.

What you need to begin:

- Any code editor (we will be demonstrating using Visual Studio Code)
- .NET Core 3.1 on your development machine. You can download this version of the .NET Core SDK for multiple platforms from Download .NET Core 3.1.

### 1.1.1. Set up local Azure credentials

1. Open Visual Studio Code
2. Verify, or if not installed, Install Azure Account extension
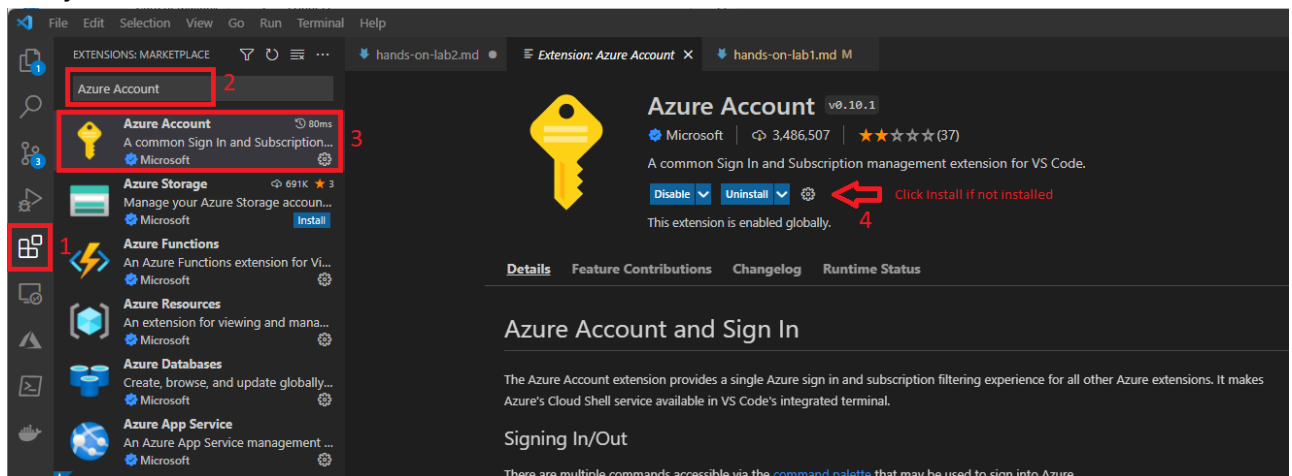


3. Click "View"->"Command Palette...", then type "Sign In" to find the correct command. Click "Azure: Sign In to Azure Cloud", then sign in to the correct account.
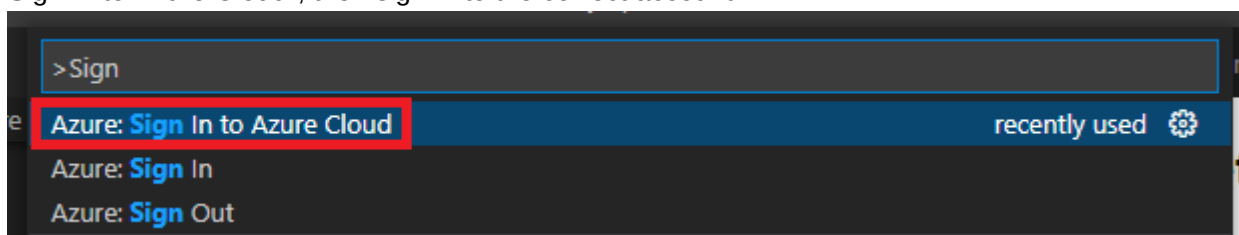


### 1.1.2. Create or reset Azure Digital Twins resource

Please refer to Section 1.2.3 of Lab 1 for instructions on how to create your Azure Digital Twins resource if it does not already exist.

If the resource already exists, open a cloud shell, then find the name of the resource

```
az dt list -o table
```

```
eric@Azure:~$ az dt list -o table
CreatedTime                     HostName                                      LastUpdatedTime                 Location       Name
------------------------------  --------------------------------------------  ------------------------------  -------------  -------------
2022-05-23T17:58:30.248986+00:00  adt-ericj0525.api.wcus.digitaltwins.azure.net  2022-05-23T17:59:21.306723+00:00  westcentralus  adt-ericj0525
```

With the name of the resource, run the following command, replacing the name with the name from the above table

```
az dt reset --dt-name [name] --yes
```

```
eric@Azure:~$ az dt reset --dt-name adt-ericj0525 --yes
This command is in preview and under development. Reference and support levels: https://aka.ms/CLI_refstatus
Found 4 twin(s).
Found 1 relationship(s) associated with twin Room1.
Found 1 relationship(s) associated with twin Floor0.
Found 0 relationship(s) associated with twin Floor1.
Found 0 relationship(s) associated with twin Room0.
eric@Azure:~$
```

## 1.2. Project Code

### 1.2.1. Create project and add dependencies

Open a new terminal in VS Code and create an empty project directory where you want to store your work during this lab. Name the directory whatever you want (for example, DigitalTwinsCodeLab).

Navigate into the new directory.

Once in the project directory, create an empty .NET console app project. In the command window, you can run the following command to create a minimal C# project for the console:

```
dotnet new console
```

This command will create several files inside your directory, including one called **Program.cs** where you'll write most of your code.

Next, add two dependencies to your project that will be needed to work with Azure Digital Twins. The first is the package for the Azure Digital Twins SDK for .NET, the second provides tools to help with authentication against Azure.

```
dotnet add package Azure.DigitalTwins.Core
dotnet add package Azure.Identity
```

### 1.2.2. Get started with project code

In this section, you'll begin writing the code for your new app project to work with Azure Digital Twins. The actions covered include:

- Authenticating against the service

- Uploading a model
- Catching errors
- Creating digital twins
- Creating relationships
- Querying digital twins

There's also a section showing the complete code at the end of the lab. You can use this section as a reference to check your program as you go.

To begin, open the file **_Program.cs_** in any code editor. You'll see a minimal code template that looks something like this:

```
1    // See https://aka.ms/new-console-template for more information
2    Console.WriteLine("Hello, World!");
```

First, replace the existing code with the following to pull in necessary dependencies

```
using Azure.DigitalTwins.Core;
using Azure.Identity;
```

Next, add code to this file to enable some functionality.

## 1.2.3. Authenticating against the service

The first thing your app will need to do is authenticate against the Azure Digital Twins service. Then, you can create a service client class to access the SDK functions.

To authenticate, you need the host name of your Azure Digital Twins instance.

In **_Program.cs_**, paste the following code below the "Hello, World!" printout line in the Main method. Set the value of adtInstanceUrl to your Azure Digital Twins instance host name.

```
string adtInstanceUrl = "https://<your-Azure-Digital-Twins-host-name>";

var credential = new DefaultAzureCredential();
var client = new DigitalTwinsClient(new Uri(adtInstanceUrl), credential);
Console.WriteLine($"Service client created – ready to go");
```

Save the file.

In your command window, run the code with this command:

```
dotnet run
```

This command will restore the dependencies on first run, then execute the program.

- If no error occurs, the program will print: "Service client created - ready to go".
- Since there isn't yet any error handling in this project, if there are any issues, you'll see an exception thrown by the code. Please verify that it was copied correctly and that your Azure Digital Twins Host name was copied into the correct spot in the code.

# 1.3. Create a model

Azure Digital Twins has no intrinsic domain vocabulary. The types of elements in your environment that you can represent in Azure Digital Twins are defined by you, using models. Models are similar to classes in object-oriented programming languages; they provide user-defined templates for digital twins to follow and instantiate later. They're written in a JSON-like language called Digital Twins Definition Language (DTDL).

The first step in creating an Azure Digital Twins solution is defining at least one model in a DTDL file.

In the directory where you created your project, create a new .json file called **SampleModel.json**. Paste in the following file body:

```json
{
  "@id": "dtmi:example:SampleModel;1",
  "@type": "Interface",
  "displayName": "SampleModel",
  "contents": [
    {
      "@type": "Relationship",
      "name": "contains"
    },
    {
      "@type": "Property",
      "name": "data",
      "schema": "string"
    }
  ],
  "@context": "dtmi:dtdl:context;2"
}
```

## 1.3.1. Upload the model

Next, add some more code to **Program.cs** to upload the model you've created into your Azure Digital Twins instance.

First, add a few using statements to the top of the file:

```csharp
using System.IO;
using System.Collections.Generic;
using Azure;
```

Next comes the first bit of code that interacts with the Azure Digital Twins service. This code loads the DTDL file you created from your disk, and then uploads it to your Azure Digital Twins service instance.

Paste in the following code under the authorization code you added earlier.

```
Console.WriteLine();
Console.WriteLine($"Upload a model");
string dtdl = File.ReadAllText("SampleModel.json");
var models = new List<string> { dtdl };
// Upload the model to the service
client.CreateModels(models);
```

In your command window, run the program with this command:

```
dotnet run
```

"Upload a model" will be printed in the output, indicating that this code was reached, but there's no output yet to indicate whether the upload was successful.

To add a print statement showing all models that have been successfully uploaded to the instance, add the following code right after the previous section:

```
foreach (DigitalTwinsModelData md in client.GetModels())
{
    Console.WriteLine($"Model: {md.Id}");
}
```

Before you run the program again to test this new code, recall that the last time you ran the program, you uploaded your model already. Azure Digital Twins won't let you upload the same model twice, so if you attempt to upload the same model again, the program should throw an exception.

With this information in mind, run the program again with this command in your command window:

```
dotnet run
```

The program should throw an exception. When you attempt to upload a model that has been uploaded already, the service returns a "bad request" error via the REST API. As a result, the Azure Digital Twins client SDK will in turn throw an exception, for every service return code other than success.

The next section talks about exceptions like this and how to handle them in your code.

## 1.3.2. Catching errors

To keep the program from crashing, you can add exception code around the model upload code. Wrap the existing client call "client.CreateModels(typeList)" in a try/catch handler, like this:

```
try
{
    client.CreateModels(models);
    Console.WriteLine("Models uploaded to the instance:");
}
catch (RequestFailedException e)
{
    Console.WriteLine($"Upload model error: {e.Status}: {e.Message}");
}
```

Run the program again with dotnet run in your command window. You'll see that you get back more details about the model upload issue, including an error code stating that ModelIdAlreadyExists.

From this point forward, the lab will wrap all calls to service methods in try/catch handlers.

## 1.4. Create digital twins

Now that you've uploaded a model to Azure Digital Twins, you can use this model definition to create digital twins. Digital twins are instances of a model, and represent the entities within your business environment—things like sensors on a farm, rooms in a building, or lights in a car. This section creates a few digital twins based on the model you uploaded earlier.

Add the following code to the end of the Main method to create and initialize three digital twins based on this model.

```
var twinData = new BasicDigitalTwin();
twinData.Metadata.ModelId = "dtmi:example:SampleModel;1";
twinData.Contents.Add("data", $"Hello World!");

string prefix = "sampleTwin-";
for (int i = 0; i < 3; i++)
{
    try
    {
        twinData.Id = $"{prefix}{i}";
        client.CreateOrReplaceDigitalTwin<BasicDigitalTwin>(twinData.Id,
twinData);
        Console.WriteLine($"Created twin: {twinData.Id}");
    }
    catch(RequestFailedException e)
    {
        Console.WriteLine($"Create twin error: {e.Status}: {e.Message}");
    }
}
```

## 1.5. Create relationships

Next, you can create relationships between the twins you've created, to connect them into a twin graph. Twin graphs are used to represent your entire environment.

Add a new static method to the Program class underneath the Main method (the code now has two methods):

```csharp
void CreateRelationship(DigitalTwinsClient client, string srcId, string targetId)
{
    var relationship = new BasicRelationship
    {
        TargetId = targetId,
        Name = "contains"
    };

    try
    {
        string relId = $"{srcId}-contains->{targetId}";
        client.CreateOrReplaceRelationship(srcId, relId, relationship);
        Console.WriteLine("Created relationship successfully");
    }
    catch (RequestFailedException e)
    {
        Console.WriteLine($"Create relationship error: {e.Status}: {e.Message}");
    }
}
```

Next, add the following code to the end of the Main method, to call the CreateRelationship method and use the code you just wrote:

```csharp
// Connect the twins with relationships
CreateRelationship(client, "sampleTwin-0", "sampleTwin-1");
CreateRelationship(client, "sampleTwin-0", "sampleTwin-2");
```

In your command window, run the program with `dotnet run`. In the output, look for print statements saying that the two relationships were created successfully.

Azure Digital Twins won't let you create a relationship if another relationship with the same ID already exists—so if you run the program multiple times, you'll see exceptions on relationship creation. This code catches the exceptions and ignores them.

## 1.5.1. List relationships

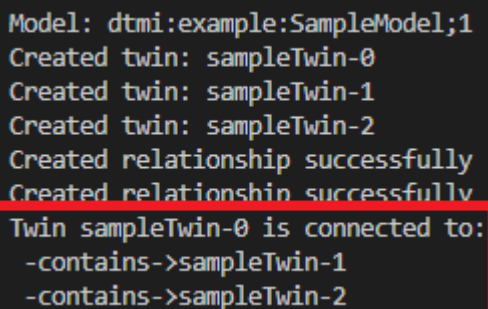The next code you'll add allows you to see the list of relationships you've created.

Add the following new method to the Program class:

```csharp
void ListRelationships(DigitalTwinsClient client, string srcId)
{
    try
    {
        Pageable<BasicRelationship> results =
client.GetRelationships<BasicRelationship>(srcId);
        Console.WriteLine($"Twin {srcId} is connected to:");
        foreach (BasicRelationship rel in results)
        {
            Console.WriteLine($" -{rel.Name}->{rel.TargetId}");
        }
    }
    catch (RequestFailedException e)
    {
        Console.WriteLine($"Relationship retrieval error: {e.Status}:
{e.Message}");
    }
}
```

Then, add the following code to the end of the Main method to call the ListRelationships code:

```csharp
//List the relationships
ListRelationships(client, "sampleTwin-0");
```

In your command window, run the program with `dotnet run`. You should see a list of all the relationships you've created in an output statement that looks like this:



## 1.6. Query digital twins

A main feature of Azure Digital Twins is the ability to query your twin graph easily and efficiently to answer questions about your environment.

The last section of code to add in this tutorial runs a query against the Azure Digital Twins instance. The query used in this example returns all the digital twins in the instance.

Add this `using` statement to enable use of the `JsonSerializer` class to help present the digital twin information:

```
using System.Text.Json;
```

Then, add the following code to the end of the `Main` method:

```csharp
// Run a query for all twins
string query = "SELECT * FROM digitaltwins";
Pageable<BasicDigitalTwin> queryResult = client.Query<BasicDigitalTwin>
(query);

foreach (BasicDigitalTwin twin in queryResult)
{
    Console.WriteLine(JsonSerializer.Serialize(twin));
    Console.WriteLine("--------------");
}
```

In your command window, run the program with `dotnet run`. You should see all the digital twins in this instance in the output.

## 1.7. Recap

At this point in the lab, you have a complete client app that can perform basic actions against Azure Digital Twins. For reference, the full code of the program in **_Program.cs_** should look similar to the code below:

```csharp
using System;
using System.IO;
using System.Collections.Generic;
using Azure;
using Azure.DigitalTwins.Core;
using Azure.Identity;
using System.Text.Json;

string adtInstanceUrl = "https://adt-
ericj0525.api.wcus.digitaltwins.azure.net";

var credential = new DefaultAzureCredential();
var client = new DigitalTwinsClient(new Uri(adtInstanceUrl), credential);
Console.WriteLine($"Service client created – ready to go");
Console.WriteLine();
Console.WriteLine($"Upload a model");
string dtdl = File.ReadAllText("SampleModel.json");
var models = new List<string> { dtdl };
// Upload the model to the service
try
{
    client.CreateModels(models);
    Console.WriteLine("Models uploaded to the instance:");
}
catch (RequestFailedException e)
```

```csharp
{
    Console.WriteLine($"Upload model error: {e.Status}: {e.Message}");
}

foreach (DigitalTwinsModelData md in client.GetModels())
{
    Console.WriteLine($"Model: {md.Id}");
}

var twinData = new BasicDigitalTwin();
twinData.Metadata.ModelId = "dtmi:example:SampleModel;1";
twinData.Contents.Add("data", $"Hello World!");

string prefix = "sampleTwin-";
for (int i = 0; i < 3; i++)
{
    try
    {
        twinData.Id = $"{prefix}{i}";
        client.CreateOrReplaceDigitalTwin<BasicDigitalTwin>(twinData.Id,
twinData);
        Console.WriteLine($"Created twin: {twinData.Id}");
    }
    catch(RequestFailedException e)
    {
        Console.WriteLine($"Create twin error: {e.Status}: {e.Message}");
    }
}

// Connect the twins with relationships
CreateRelationship(client, "sampleTwin-0", "sampleTwin-1");
CreateRelationship(client, "sampleTwin-0", "sampleTwin-2");

//List the relationships
ListRelationships(client, "sampleTwin-0");

// Run a query for all twins
string query = "SELECT * FROM digitaltwins";
Pageable<BasicDigitalTwin> queryResult = client.Query<BasicDigitalTwin>
(query);

foreach (BasicDigitalTwin twin in queryResult)
{
    Console.WriteLine(JsonSerializer.Serialize(twin));
    Console.WriteLine("--------------");
}

void ListRelationships(DigitalTwinsClient client, string srcId)
{
    try
    {
        Pageable<BasicRelationship> results =
client.GetRelationships<BasicRelationship>(srcId);
        Console.WriteLine($"Twin {srcId} is connected to:");
```

```csharp
            foreach (BasicRelationship rel in results)
            {
                Console.WriteLine($" -{rel.Name}->{rel.TargetId}");
            }
        }
        catch (RequestFailedException e)
        {
            Console.WriteLine($"Relationship retrieval error: {e.Status}:
{e.Message}");
        }
    }

    void CreateRelationship(DigitalTwinsClient client, string srcId, string
targetId)
    {
        var relationship = new BasicRelationship
        {
            TargetId = targetId,
            Name = "contains"
        };

        try
        {
            string relId = $"{srcId}-contains->{targetId}";
            client.CreateOrReplaceRelationship(srcId, relId, relationship);
            Console.WriteLine("Created relationship successfully");
        }
        catch (RequestFailedException e)
        {
            Console.WriteLine($"Create relationship error: {e.Status}:
{e.Message}");
        }
    }
```

## 1.8. Clean up

After completing this lab, please move on to Lab Three.

If you would like to reset your Azure Digital Twins prior to the next lab, please go to section 1.1.1 and follow the directions there.

You may also remove your resource group and the Azure Digital Twins resource within it using:

```
az group delete --name <your-resource-group>
```