Experiment: Data pipelines using Azure Blob storage, Azure SQL Database, and Azure Data Factory

In this experiment, you create a data factory by using the Azure Data Factory user interface (UI). The pipeline in this data factory copies data from Azure Blob storage to a database in Azure SQL Database. The configuration pattern in this experiment applies to copying from a file-based data store to a relational data store. For a list of data stores supported as sources and sinks, see the supported data stores table.

Note

Prerequisites

- **Azure subscription**. If you don't have an Azure subscription, create a free Azure account before you begin.
- **Azure storage account**. You use Blob storage as a *source* data store. If you don't have a storage account, see Create an Azure storage account for steps to create one.
- **Azure SQL Database**. You use the database as a *sink* data store. If you don't have a database in Azure SQL Database, see the Create a database in Azure SQL Database for steps to create one.

How to do it…

The steps for this experiment are as follows:

1. Execute the following commands to create the employee container in our Azure storage account:

```
$storageaccountname="adestoragepowershellxx"
$containername="input"
$resourcegroup="AzureDataEngineeringxx"
#Get the Azure Storage account context
    $storagecontext = (Get-AzStorageAccount -ResourceGroupName
$resourcegroup -Name $storageaccountname).Context;
#Create a new container
New-AzStorageContainer -Name $containername -Context
$storagecontext
```

Container creation is usually very quick.

Note: Substitute for the student number for xx, (student1 would be adestoragepowershell01, student12 would be adestoragepowershell12). Similarly as we did earlier for the AzureDataEngineeringxx ResourceGroupName (student1 would be AzureDataEngineering01, student12 would be AzureDataEngineering12).

```
PS C:\projects\AzureDataEngineering> #Create a new container
>> New-AzStorageContainer -Name $containername -Context $storagecontext

   Storage Account Name: adestoragepowershell

Name                 PublicAccess         LastModified                  IsDeleted  VersionId
----                 ------------         ------------                  ---------  ---------
input                Off                  5/19/2022 12:35:22 PM +00:00
```

Create a blob and a SQL table

Create a source blob

1. Launch Notepad. Copy the following text, and save it as an **emp.txt** file on your disk:

   Copy
   FirstName,LastName
   John,Doe
   Jane,Doe

2. We can use CLI or the Azure Storage Explorer to upload this file to our input container created earlier in the experience.

Set-AzStorageBlobContent -File "C:\projects\AzureDataEngineering\repos\azure-data-engineering\labs\emp.txt" -Context $storagecontext -Blob emp.txt -Container $containername

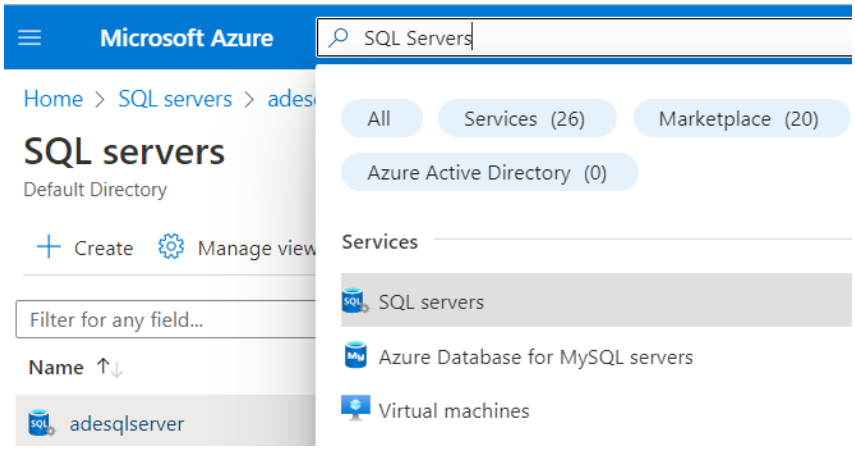
```
PS C:\projects\AzureDataEngineering> Set-AzStorageBlobContent -File "C:\projects\AzureDataEngineering\rep
context -Blob emp.txt  -Container $containername

   AccountName: adestoragepowershell, ContainerName: input

Name                 BlobType  Length          ContentType                    LastModified         Access
----                 --------  ------          -----------                    ------------         ------
emp.txt              BlockBlob 38              application/octet-stream       2022-05-19 12:36:20Z Hot
```

Create a sink SQL table

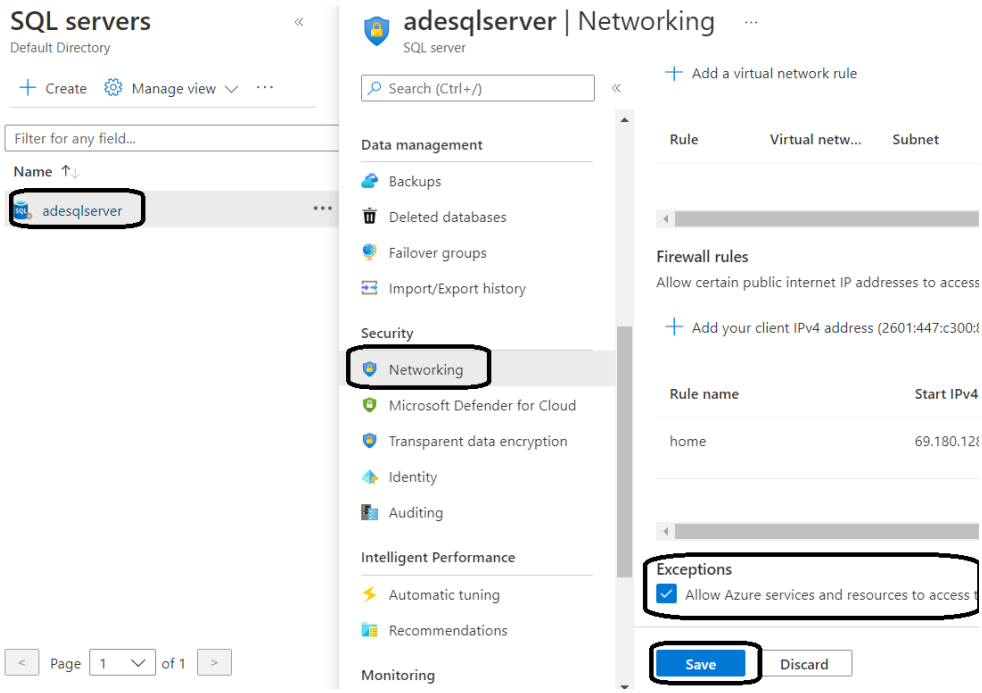
1. The SQL Server PowerShell module and our Azure Synapse SQL Pool were created in an earlier experiment

```
Invoke-Sqlcmd -ServerInstance "adesqlserverxx.database.windows.net" -U
sqladmin -P "Sql@Server@1234" -Database adesqldwxx  -InputFile
C:\projects\AzureDataEngineering\repos\azure-data-
engineering\labs\Emp.sql
```

2. Find your SQL Server instance that was created in our previous experiment.



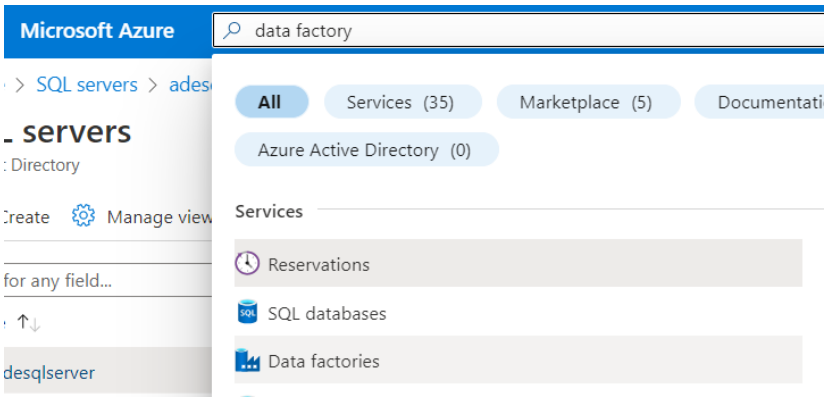
3. Allow Azure services to access for our SQL Server **adesqldwxx** instance. Ensure that **Allow access to Azure services** is turned **ON** for your SQL Server so that Data Factory can write data to your SQL Server. To verify and turn on this setting, go to logical SQL server > Overview > Set server firewall> set the **Allow access to Azure services** option to **ON**.



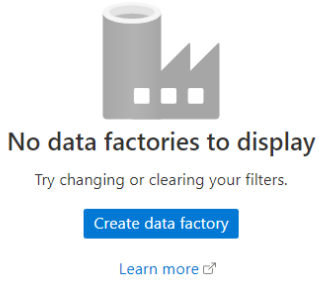
Create a data factory

In this step, you create a data factory and start the Data Factory UI to create a pipeline in the data factory.

1. Open **Microsoft Edge** or **Google Chrome**. Currently, Data Factory UI is supported only in Microsoft Edge and Google Chrome web browsers. Use the search dialog to open the **Data Factories** blade.



2. Select **Create data factory**.

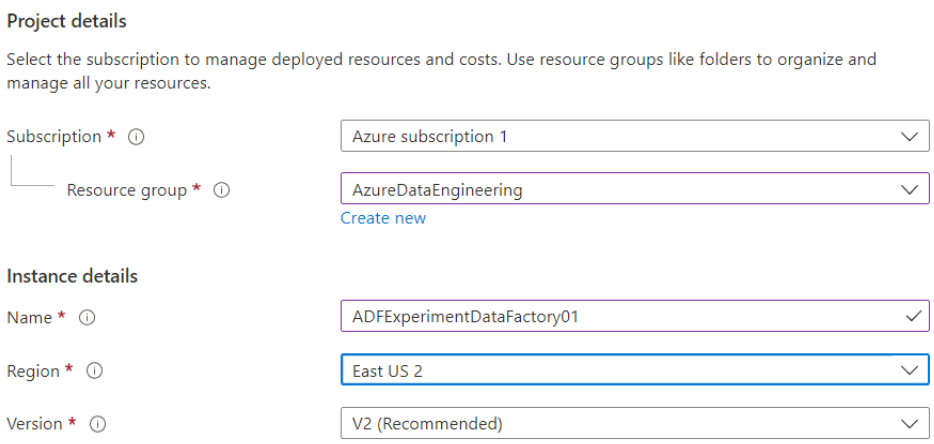


3. On the **Create Data Factory** page, under **Basics** tab, our default Azure **Subscription** should be select.
4. For **Resource Group**, take one of the following steps:

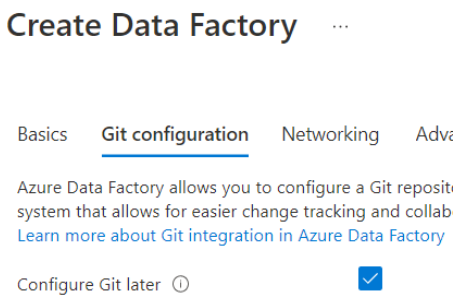
   a. Select our existing **AzureDataEngineeringxx** resource group from the drop-down list.

5. Under **Region**, select the **East US 2** location for the data factory. Only locations that are supported are displayed in the drop-down list. The data stores (for example, Azure Storage and SQL Database) and computes (for example, Azure HDInsight) used by the data factory can be in other regions. We'll use **East US 2**.
6. Under **Name**, enter **ADFExperimentDataFactoryxx**. Remember to substitute your student number for the **xx**, for example **student1** would use **ADFExperimentDataFactory01**

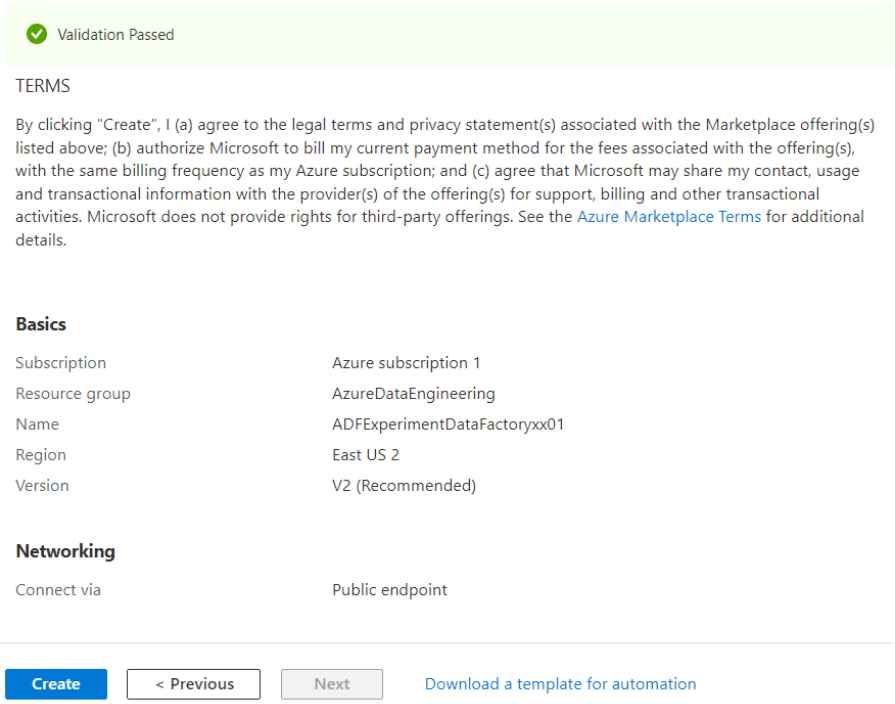
   The name of the Azure data factory must be *globally unique*. If you receive an error message about the name value, enter a different name for the data factory. (for example, ADFExperimentDataFactory01). For naming rules for Data Factory artifacts, see Data Factory naming rules.



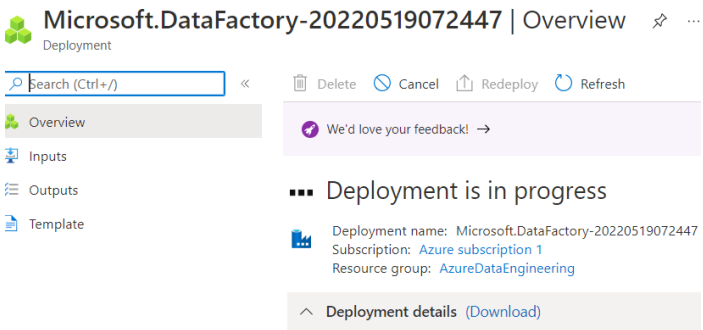
7. Under **Version**, select **V2**.
8. Select **Git configuration** tab on the top, and select the **Configure Git later** check box.



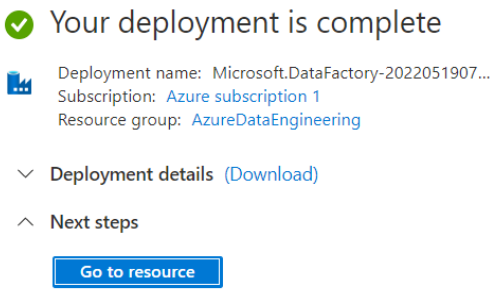
9. Select **Review + create**



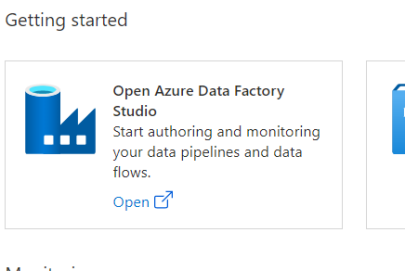
10. **S**elect **Create** after the validation is passed.



11. After the creation is finished, you see the notice in Notifications center. Select **Go to resource** to navigate to the Data factory page.



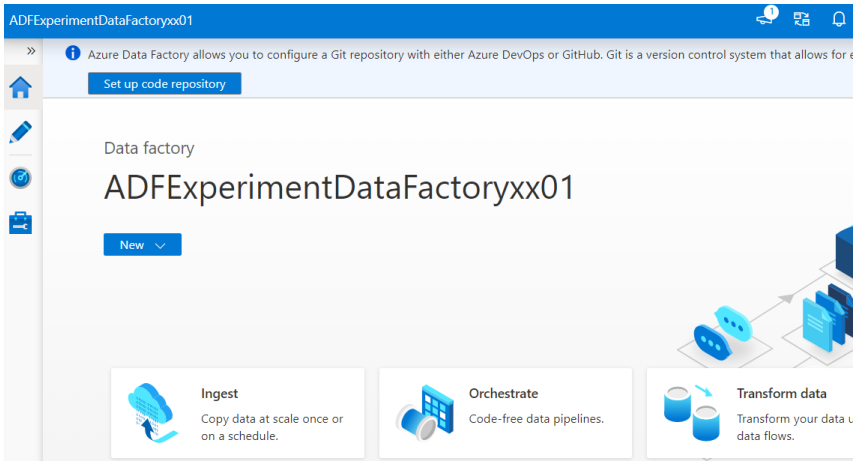
12. Select **Open** on the **Open Azure Data Factory Studio** tile to launch the Azure Data Factory UI in a separate tab.



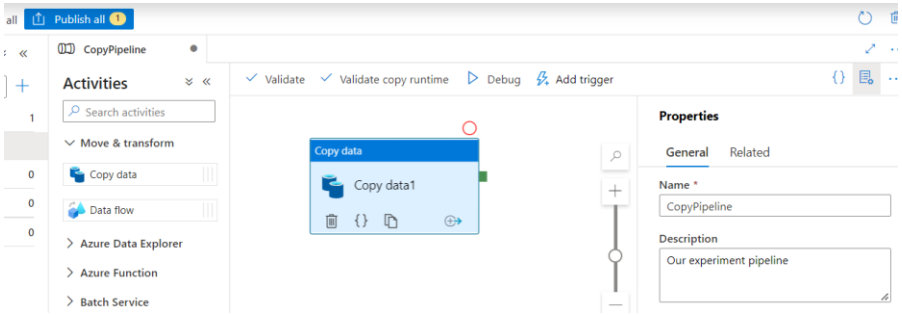
Create a pipeline

In this step, you create a pipeline with a copy activity in the data factory. The copy activity copies data from Blob storage to SQL Database.

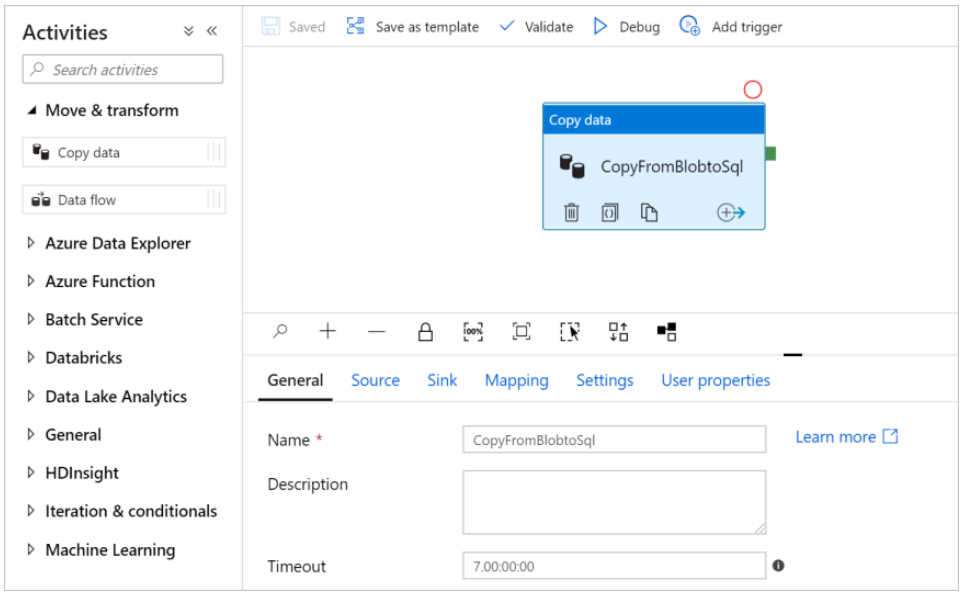
1. On the home page, select **Orchestrate**.



1. In the General panel under **Properties**, specify **CopyPipeline** for **Name**. Then collapse the panel by clicking the Properties icon in the top-right corner.

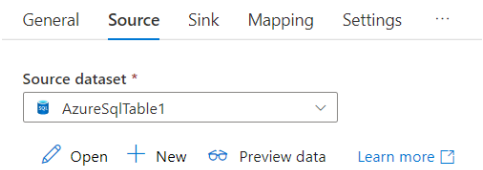


2. In the **Activities** tool box, expand the **Move and Transform** category, and drag and drop the **Copy Data** activity from the tool box to the pipeline designer surface. Specify **CopyFromBlobToSql** for **Name**.

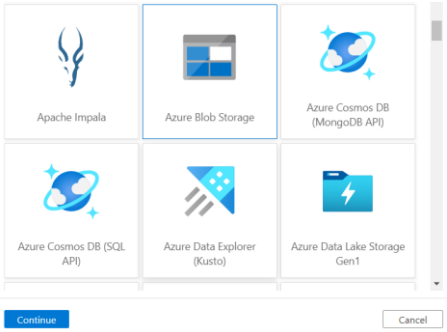


Configure source

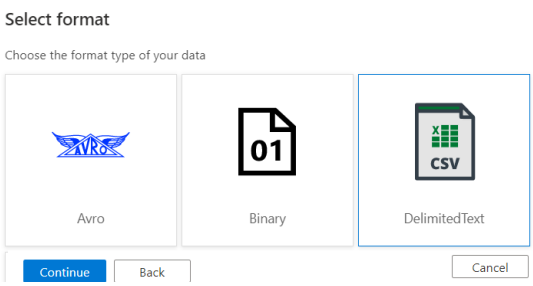
1. Go to the **Source** tab. Select **+ New** to create a source dataset.



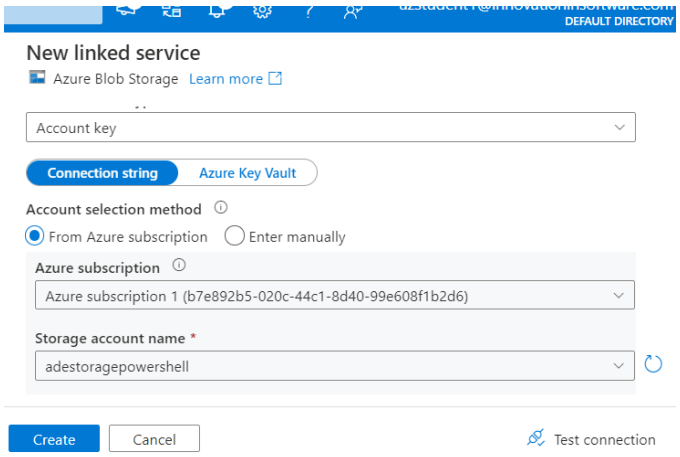
2. In the **New Dataset** dialog box, select **Azure Blob Storage**, and then select **Continue**. The source data is in Blob storage, so you select **Azure Blob Storage** for the source dataset.



3. In the **Select Format** dialog box, choose the format type of your data, and then select **Continue**.

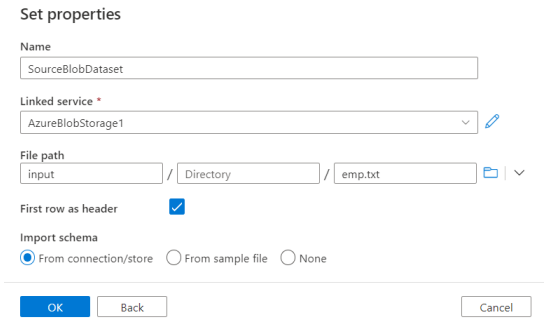


4. In the **Set Properties** dialog box, enter **SourceBlobDataset** for Name. Select the checkbox for **First row as header**. Under the **Linked service** text box, select **+ New**.

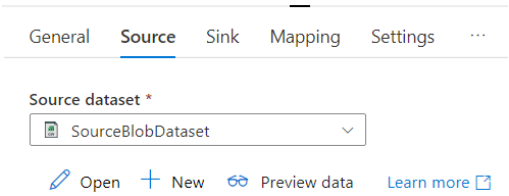


5. In the **New Linked Service (Azure Blob Storage)** dialog box, enter **AzureStorageLinkedService** as name, select your storage account from the **Storage account name** list.

6. Test connection, select **Create** to deploy the linked service.
7. After the linked service is created, it's navigated back to the **Set properties** page. Next to **File path**, select the folder icon next to the folder/file input boxes.

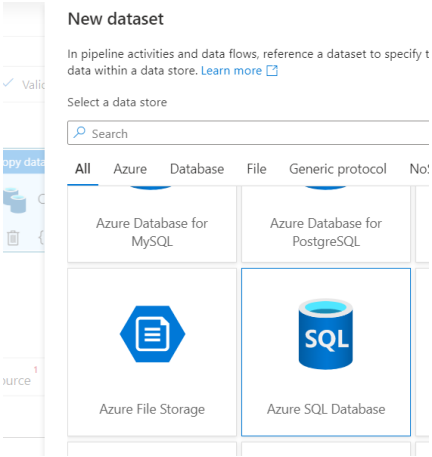


8. Navigate to the **input** folder, select the **emp.txt** file, and then select **OK**.
9. Select **OK**. It automatically navigates to the pipeline page. In **Source** tab, confirm that **SourceBlobDataset** is selected. To preview data on this page, select **Preview data**.

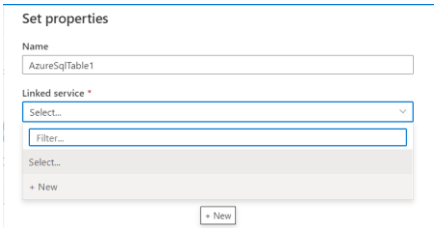


Configure sink

1. Go to the **Sink** tab, and select **+ New** to create a sink dataset.
2. In the **New Dataset** dialog box, input "SQL" in the search box to filter the connectors, select **Azure SQL Database**, and then select **Continue**. In this experiment, you copy data to a SQL database.



3. In the **Set Properties** dialog box, enter **OutputSqlDataset** for Name. From the **Linked service** dropdown list, select **+ New**.



4. A dataset must be associated with a linked service. The linked service has the connection string that Data Factory uses to connect to SQL Database at runtime. The dataset specifies the container, folder, and the file (optional) to which the data is copied.
5. In the **New Linked Service (Azure SQL Database)** dialog box, take the following steps:

a. Under **Name**, enter **AzureSqlDatabaseLinkedService**.

b. Under **Server name**, select your SQL Server instance.

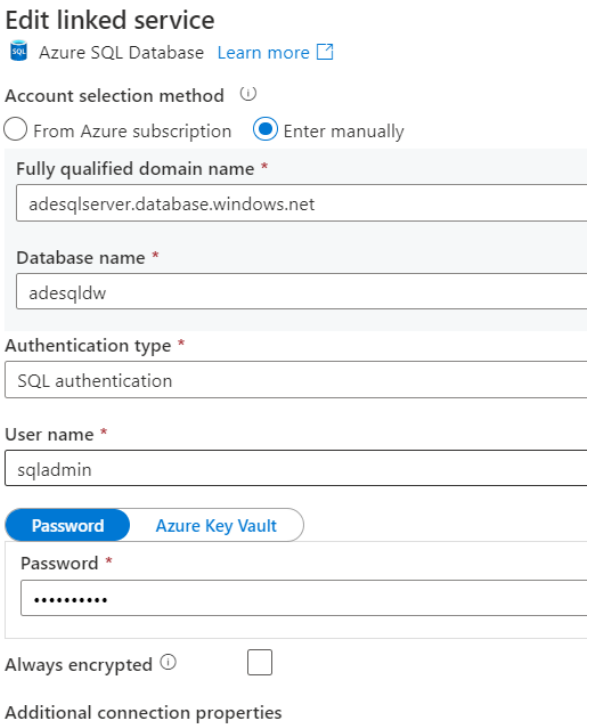
c. Under **Database name**, select your **adesqldwxx**.

d. Under **User name**, enter the name of the user.

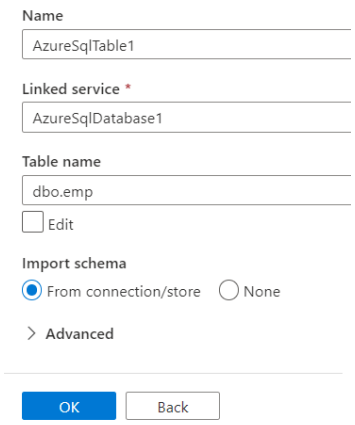
e. Under **Password**, enter the password for the user.

f. Select **Test connection** to test the connection.

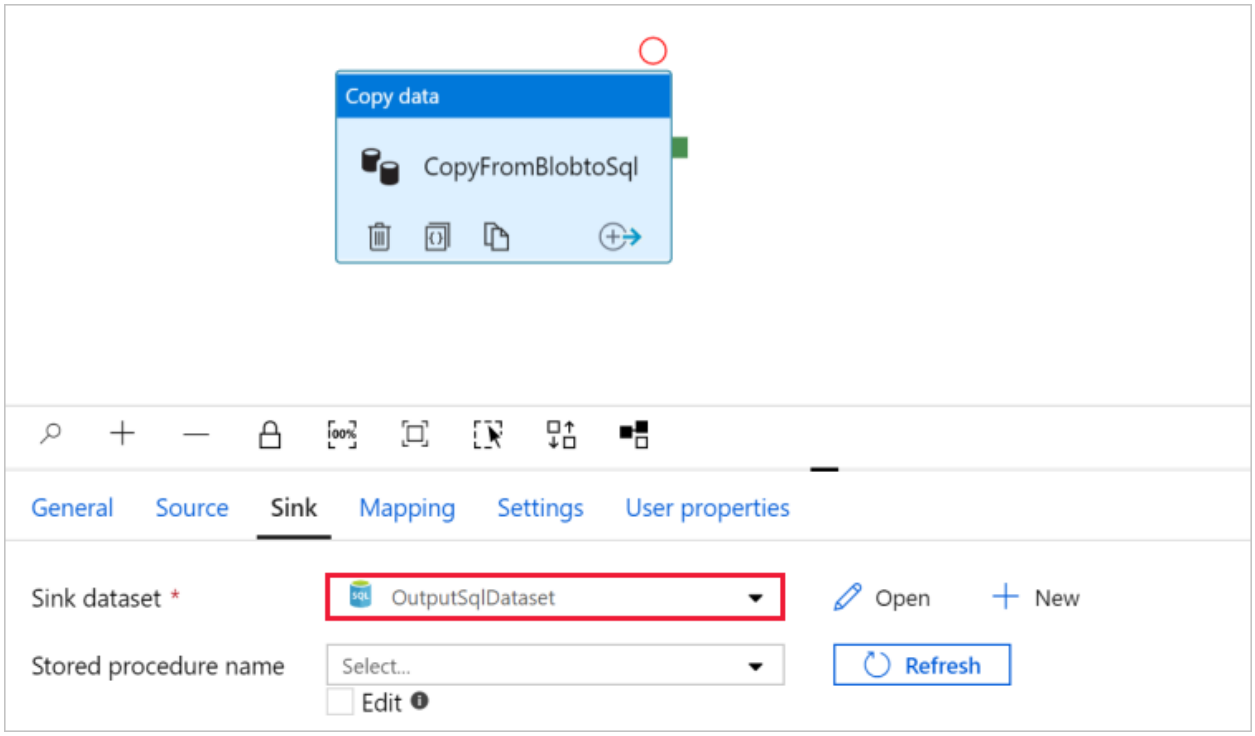
g. Select **Create** to deploy the linked service.



6. It automatically navigates to the **Set Properties** dialog box. In **Table**, select **[dbo].[emp]**. Then select **OK**.



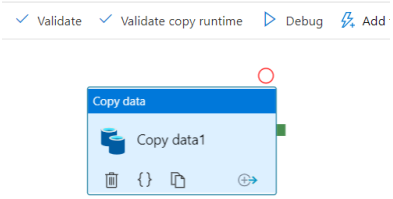
7. Go to the tab with the pipeline, and in **Sink Dataset**, confirm that **OutputSqlDataset** is selected.



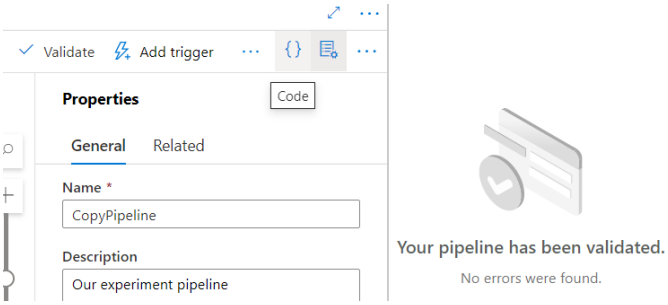
You can optionally map the schema of the source to corresponding schema of destination by following Schema mapping in copy activity.

Validate the pipeline

To validate the pipeline, select **Validate** from the tool bar.



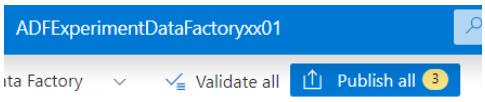
You can see the JSON code associated with the pipeline by clicking **Code** on the upper right.



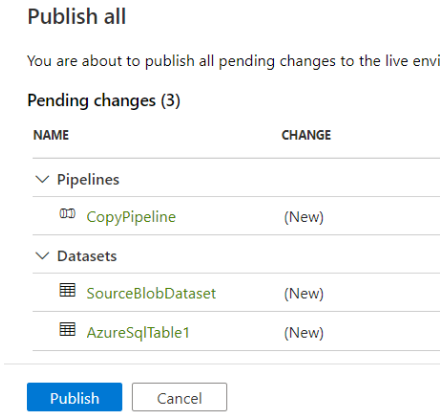
Debug and publish the pipeline

You can debug a pipeline before you publish artifacts (linked services, datasets, and pipeline) to Data Factory or your own Azure Repos Git repository.

1. To debug the pipeline, select **Debug** on the toolbar. You see the status of the pipeline run in the **Output** tab at the bottom of the window.
2. Once the pipeline can run successfully, in the top toolbar, select **Publish all**. This action publishes entities (datasets, and pipelines) you created to Data Factory.



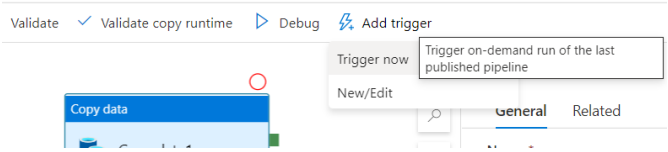
3. Wait until you see the **Successfully published** message. To see notification messages, click the **Show Notifications** on the top-right (bell button).



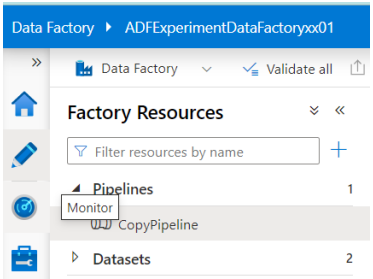
Trigger the pipeline manually

In this step, you manually trigger the pipeline you published in the previous step.

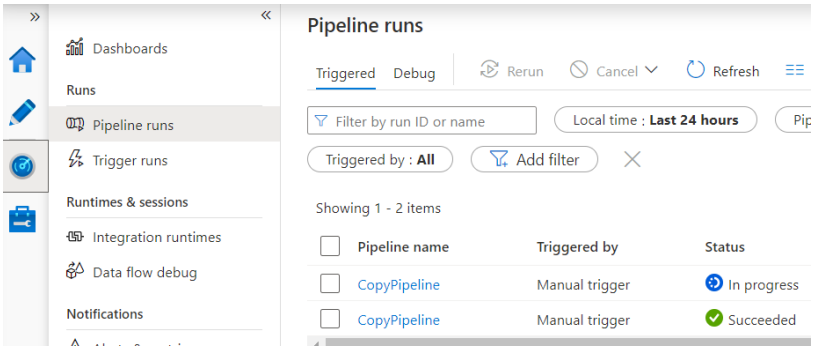
1. Select **Trigger** on the toolbar, and then select **Trigger Now**. On the **Pipeline Run** page, select **OK**.



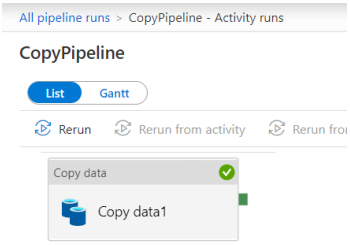
2. Go to the **Monitor** tab on the left.



3. You see a pipeline run that is triggered by a manual trigger. You can use links under the **PIPELINE NAME** column to view activity details and to rerun the pipeline.



4. To see activity runs associated with the pipeline run, select the **CopyPipeline** link under the **PIPELINE NAME** column. In this example, there's only one activity, so you see only one entry in the list. For details about the copy operation, select the **Details** link (eyeglasses icon) under the **ACTIVITY NAME** column.



5. Select **All pipeline runs** at the top to go back to the Pipeline Runs view. To refresh the view, select **Refresh**.
6. Verify that two more rows are added to the **emp** table in the database.

```
Invoke-Sqlcmd -ServerInstance "adesqlserverxx.database.windows.net" -U
sqladmin -P "Sql@Server@1234" -Database adesqldwxx -Query "select *
from emp"
```

Note: Substitute the correct naming for your ServerInstance endpoint URL and database.

```
PS C:\projects\AzureDataEngineering> Invoke-Sqlcmd -ServerInstance "adesqlserver
 "select * from emp"

 ID FirstName LastName
 -- --------- --------
 22 John      Doe
 60 John      Doe
 82 Jane      Doe
120 Jane      Doe

PS C:\projects\AzureDataEngineering>
```

Trigger the pipeline on a schedule

In this schedule, you create a schedule trigger for the pipeline. The trigger runs the pipeline on the specified schedule, such as hourly or daily. Here you set the trigger to run every minute until the specified end datetime.

1. Go to the **Author** tab on the left above the monitor tab.
2. Go to your pipeline, click **Trigger** on the tool bar, and select **New/Edit**.
3. In the **Add triggers** dialog box, select **+ New** for **Choose trigger** area.
4. In the **New Trigger** window, take the following steps:

   a. Under **Name**, enter **RunEveryMinute**.

   b. Update the **Start date** for your trigger. If the date is before current datetime, the trigger will start to take effect once the change is published.

   c. Under **Time zone**, select the drop-down list.

   d. Set the **Recurrence** to **Every 1 Minute(s)**.

   e. Select the checkbox for **Specify an end date**, and update the **End On** part to be a few minutes past the current datetime. The trigger is activated only after you publish the changes. If you set it to only a couple of minutes apart, and you don't publish it by then, you don't see a trigger run.

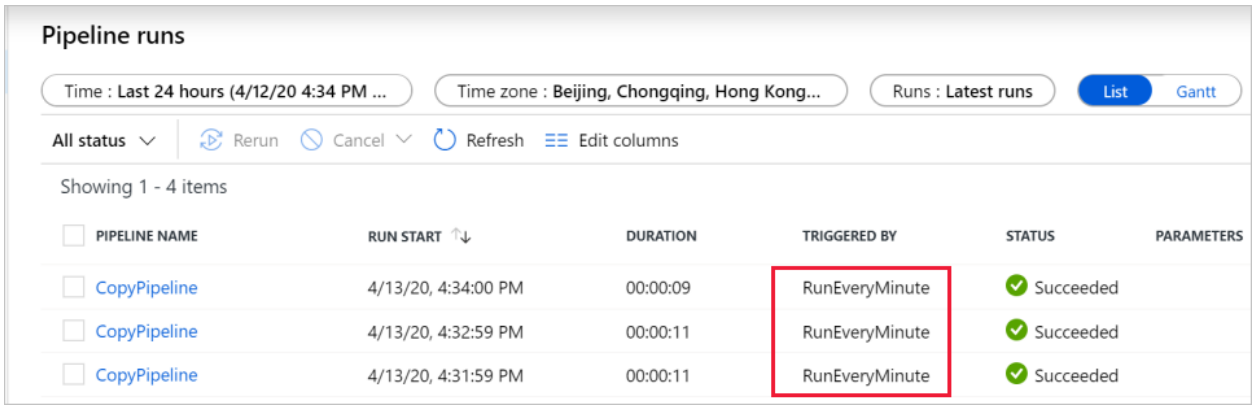
   f. For **Activated** option, select **Yes**.

g. Select **OK**.

Important

A cost is associated with each pipeline run, so set the end date appropriately.

5. On the **Edit trigger** page, review the warning, and then select **Save**. The pipeline in this example doesn't take any parameters.
6. Click **Publish all** to publish the change.
7. Go to the **Monitor** tab on the left to see the triggered pipeline runs.



8. To switch from the **Pipeline Runs** view to the **Trigger Runs** view, select **Trigger Runs** on the left side of the window.
9. You see the trigger runs in a list.
10. Verify that two rows per minute (for each pipeline run) are inserted into the **emp** table until the specified end time.