

Azure Data Factory

Introduction to Azure Data Factory

With the wide range of data stores available in Azure, there is the need to manage and orchestrate the movement data between them. In fact, you may want to automate a regular process of data movement as part of a wider enterprise analytical solution. Both Azure Data Factory and Azure Synapse Pipelines meet that need, and in this section, you will be introduced to this technology, its component parts, the Azure Data Factory process, and the security required to provision and manage the service.

Understand Azure Data Factory

The need to trigger the batch movement of data, or to set up a regular schedule is a requirement for most analytics solutions. Azure Data Factory (ADF) is the service that can be used to fulfill such a requirement. ADF provides a cloud-based data integration service that orchestrates the movement and transformation of data between various data stores and compute resources.

Azure Data Factory is the cloud-based ETL and data integration service that allows you to create data-driven workflows for orchestrating data movement and transforming data at scale. Using Azure Data Factory, you can create and schedule data-driven workflows (called pipelines) that can ingest data from disparate data stores. You can build complex ETL processes that transform data visually with data flows or by using compute services such as Azure HDInsight Hadoop, Azure Databricks, and Azure Synapse Analytics.

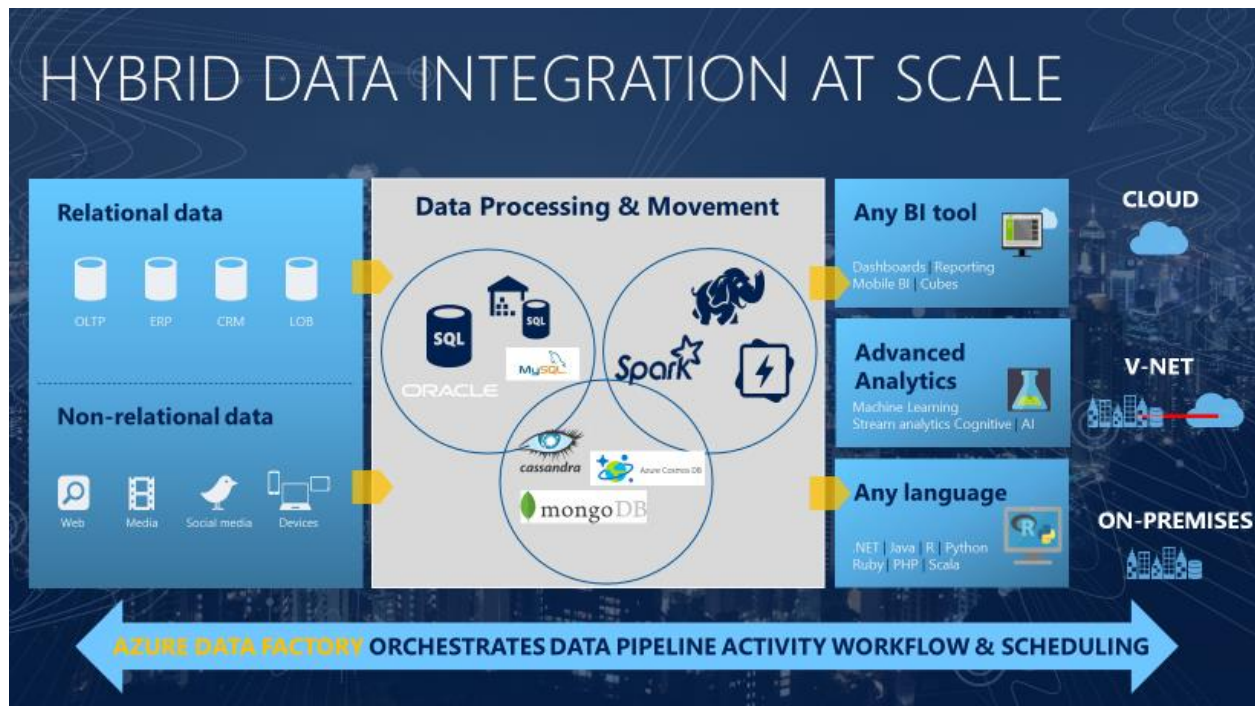
What is meant by orchestration

To use an analogy, think about a symphony orchestra. The central member of the orchestra is the conductor. The conductor does not play the instruments, they simply lead the symphony members through the entire piece of music that they perform. The musicians use their own skills to produce particular sounds at various stages of the symphony, so they may only learn certain parts of the music. The conductor orchestrates the entire piece of music, and therefore is aware of the entire score that is being performed. They will also use specific arm movements that provide instructions to the musicians how a piece of music should be played.

ADF can use a similar approach, whilst it has native functionality to ingest and transform data, sometimes it will instruct another service to perform the actual work required on its behalf, such as a Databricks to execute a transformation query. So, in this case, it would be Databricks that

performs the work, not ADF. ADF merely orchestrates the execution of the query, and then provides the pipelines to move the data onto the next step or destination.

It also provides rich visualizations to display the lineage and dependencies between your data pipelines, and monitor all your data pipelines from a single unified view to easily pinpoint issues and setup monitoring alerts.



Describe data integration patterns

Microsoft Azure provides a variety of data platform services that enables you to perform different types of analytics. Whether it is a descriptive analytics solution in a data warehouse, through to predictive analytics within HDInsight, Azure Databricks or Machine Learning Services. There is a need for a service to deal with the important aspect of data integration.

Data integration firstly involves the collection of data from one or more sources. Optionally, it typically then includes a process where the data may be cleansed and

transformed, or perhaps augmented with additional data and prepared. Finally, the amalgamated data is stored in a data platform service that handles the type of analytics that you want to perform. This process can be automated by Azure Data Factory in a pattern known as Extract, Transform and Load (ETL).

Extract

During the extraction process, data engineers define the data and its source:

- **Define the data source:** Identify source details such as the resource group, subscription, and identity information such as a key or secret.
- **Define the data:** Identify the data to be extracted. Define data by using a database query, a set of files, or an Azure Blob storage name for blob storage.

Transform

- **Define the data transformation:** Data transformation operations can include splitting, combining, deriving, adding, removing, or pivoting columns. Map fields between the data source and the data destination. You might also need to aggregate or merge data.

Load

- **Define the destination:** During a load, many Azure destinations can accept data formatted as a JavaScript Object Notation (JSON), file, or blob. You might need to write code to interact with application APIs.

Azure Data Factory offers built-in support for Azure Functions. You'll also find support for many programming languages, including Node.js, .NET, Python, and Java. Although Extensible Markup Language (XML) was common in the past, most systems have migrated to JSON because of its flexibility as a semistructured data type.

- **Start the job:** Test the ETL job in a development or test environment. Then migrate the job to a production environment to load the production system.
- **Monitor the job:** ETL operations can involve many complex processes. Set up a proactive and reactive monitoring system to provide information when things go wrong. Set up logging according to the technology that will use it.

ETL tools

As a data engineer, there are several available tools for ETL. Azure Data Factory provides nearly 100 enterprise connectors and robust resources for both code-free and code-based users to accomplish their data movement and transformation needs.

Evolution from ETL

Azure has opened the way for technologies that can handle unstructured data at an unlimited scale. This change has shifted the paradigm for loading and transforming data from ETL to extract, load, and transform (ELT).

The benefit of ELT is that you can store data in its original format, be it JSON, XML, PDF, or images. In ELT, you define the data's structure during the transformation phase, so you can use the source data in multiple downstream systems.

In an ELT process, data is extracted and loaded in its native format. This change reduces the time required to load the data into a destination system. The change also limits resource contention on the data sources.

The steps for the ELT process are the same as for the ETL process. They just follow a different order.

Another process like ELT is called extract, load, transform, and load (ELTL). The difference with ELTL is that it has a final load into a destination system.

There are two common types of data integration patterns that can be supported by Azure Data Factory.

Modern Data Warehouse workloads:

A Modern Data Warehouse is a centralized data store that provides descriptive analytics and decision support services across the whole enterprise using structured, unstructured, or streaming data sources. Data flows into the warehouse from multiple transactional systems, relational databases, and other data sources on a periodic basis. The stored data is used for historical and trend analysis reporting. The data warehouse acts as a central repository for many subject areas and contains the "single source of truth."

Azure Data factory is typically used to automate the process of extracting, transforming, and loading the data through a batch process against structured and unstructured data sources.

Advanced Analytical Workloads

You can perform advanced analytics in the form of predictive or preemptive analytics using a range of Azure data platform services. Azure Data Factory provides the integration from source systems into a Data Lake store, and can initiate compute resources such as Azure Databricks, or HDInsight to use the data to perform the advanced analytical work

Explain the data factory process

Data-driven workflows

The pipelines (data-driven workflows) in Azure Data Factory typically perform the following four steps:



Connect and collect

The first step in building an orchestration system is to define and connect all the required sources of data together, such as databases, file shares, and FTP web services. The next step is to ingest the data as needed to a centralized location for subsequent processing.

Transform and enrich

Compute services such as Databricks and Machine Learning can be used to prepare or produce transformed data on a maintainable and controlled schedule to feed production environments with cleansed and transformed data. In some instances, you may even augment the source data with additional data to aid analysis, or consolidate it through a normalization process to be used in a Machine Learning experiment as an example.

Publish

After the raw data has been refined into a business-ready consumable form from the transform and enrich phase, you can load the data into Azure Data Warehouse, Azure SQL Database, Azure Cosmos DB, or whichever analytics engine your business users can point to from their business intelligence tools

Monitor

Azure Data Factory has built-in support for pipeline monitoring via Azure Monitor, API, PowerShell, Azure Monitor logs, and health panels on the Azure portal, to monitor the scheduled activities and pipelines for success and failure rates.

Components of Azure Data Factory

Top-level concepts

An Azure subscription might have one or more Azure Data Factory instances (or data factories). Azure Data Factory is composed of below key components.

- Pipelines
- Activities
- Datasets
- Linked services
- Data Flows
- Integration Runtimes

These components work together to provide the platform on which you can compose data-driven workflows with steps to move and transform data.

Pipeline

A data factory might have one or more pipelines. A pipeline is a logical grouping of activities that performs a unit of work. Together, the activities in a pipeline perform a task. For example, a pipeline can contain a group of activities that ingests data from an Azure blob, and then runs a Hive query on an HDInsight cluster to partition the data.

The benefit of this is that the pipeline allows you to manage the activities as a set instead of managing each one individually. The activities in a pipeline can be chained together to operate sequentially, or they can operate independently in parallel.

Activity

Activities represent a processing step in a pipeline. For example, you might use a copy activity to copy data from one data store to another data store. Similarly, you might use a Hive activity, which runs a Hive query on an Azure HDInsight cluster, to transform or analyze your data. Data Factory supports three types of activities: data movement activities, data transformation.

For more details: <https://docs.microsoft.com/en-us/azure/data-factory/concepts-pipelines-activities>

Mapping data flows

Create and manage graphs of data transformation logic that you can use to transform any-sized data. You can build-up a reusable library of data transformation routines and execute those processes in a scaled-out manner from your ADF pipelines. Data Factory

will execute your logic on a Spark cluster that spins-up and spins-down when you need it. You won't ever have to manage or maintain clusters.

For more details: <https://docs.microsoft.com/en-us/azure/data-factory/concepts-data-flow-overview>

Datasets

Datasets represent data structures within the data stores, which simply point to or reference the data you want to use in your activities as inputs or outputs.

Linked services

Linked services are much like connection strings, which define the connection information that's needed for Data Factory to connect to external resources. Think of it this way: a linked service defines the connection to the data source, and a dataset represents the structure of the data. For example, an Azure Storage-linked service specifies a connection string to connect to the Azure Storage account. Additionally, an Azure blob dataset specifies the blob container and the folder that contains the data.

Linked services are used for two purposes in Data Factory:

- To represent a data store that includes, but isn't limited to, a SQL Server database, Oracle database, file share, or Azure blob storage account. For a list of supported data stores, see the copy activity article.
- To represent a compute resource that can host the execution of an activity. For example, the HDInsightHive activity runs on an HDInsight Hadoop cluster. For a list of transformation activities and supported compute environments, see the transform data article.

Integration Runtime

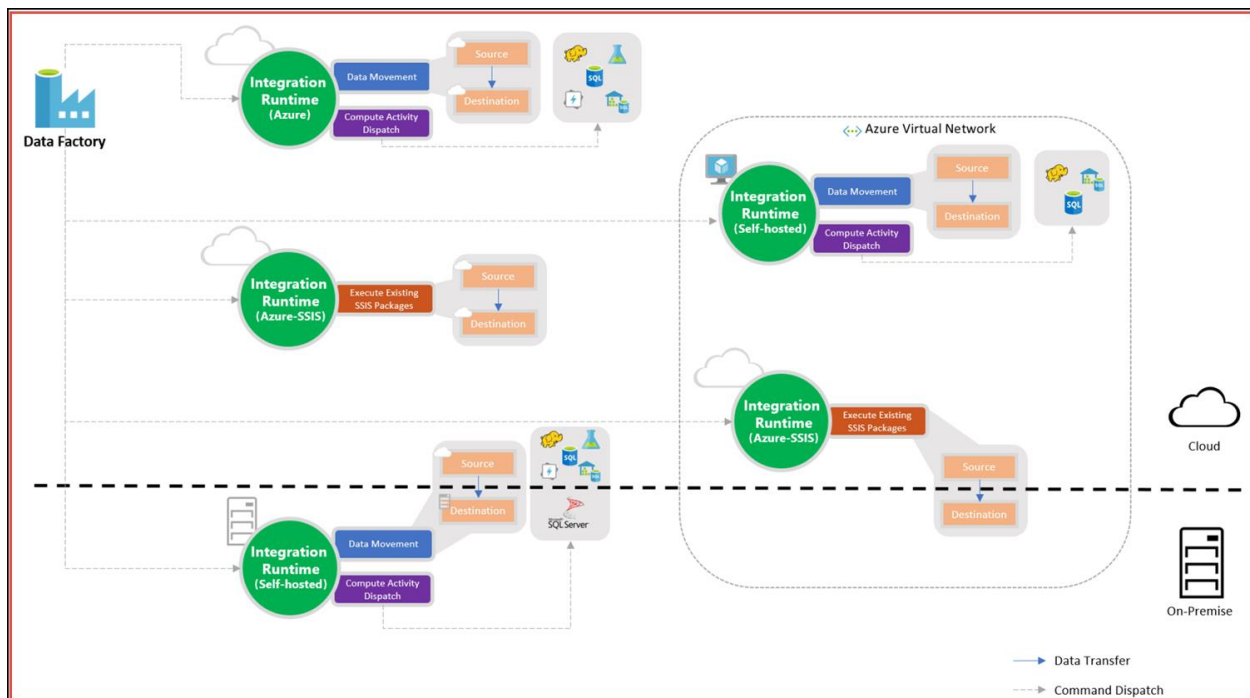
In Data Factory, an activity defines the action to be performed. A linked service defines a target data store or a compute service. An integration runtime provides the bridge between the activity and linked Services. It's referenced by the linked service or activity, and provides the compute environment where the activity either runs on or gets dispatched from. This way, the activity can be performed in the region closest possible to the target data store or compute service in the most performant way while meeting security and compliance needs.

For more details: <https://docs.microsoft.com/en-us/azure/data-factory/concepts-integration-runtime>

The following table describes the capabilities and network support for each of the integration runtime types:

INTEGRATION RUNTIME TYPES		
IR type	Public network	Private network
Azure	Data Flow	Data Flow
	Data movement	Data movement
	Activity dispatch	Activity dispatch
Self-hosted	Data movement	Data movement
	Activity dispatch	Activity dispatch
Azure-SSIS	SSIS package execution	SSIS package execution

The following diagram shows the location settings for Data Factory and its integration runtimes:



Azure integration runtime

An Azure integration runtime can:

- Run Data Flows in Azure
- Run copy activities between cloud data stores

- Dispatch the following transform activities in a public network: Databricks Notebook/ Jar/ Python activity, HDInsight Hive activity, HDInsight Pig activity, HDInsight MapReduce activity, HDInsight Spark activity, HDInsight Streaming activity, ML Studio (classic) Batch Execution activity, ML Studio (classic) Update Resource activities, Stored Procedure activity, Data Lake Analytics U-SQL activity, .NET custom activity, Web activity, Lookup activity, and Get Metadata activity.

Self-hosted integration runtime

A self-hosted IR is capable of:

- Running copy activity between a cloud data stores and a data store in private network.
- Dispatching the following transform activities against compute resources in on-premises or Azure Virtual Network: HDInsight Hive activity (BYOC-Bring Your Own Cluster), HDInsight Pig activity (BYOC), HDInsight MapReduce activity (BYOC), HDInsight Spark activity (BYOC), HDInsight Streaming activity (BYOC), ML Studio (classic) Batch Execution activity, ML Studio (classic) Update Resource activities, Stored Procedure activity, Data Lake Analytics U-SQL activity, Custom activity (runs on Azure Batch), Lookup activity, and Get Metadata activity.

Azure-SSIS integration runtime

To lift and shift existing SSIS workload, you can create an Azure-SSIS IR to natively execute SSIS packages.

Triggers

Triggers represent the unit of processing that determines when a pipeline execution needs to be kicked off. There are different types of triggers for different types of events.

Pipeline runs

A pipeline run is an instance of the pipeline execution. Pipeline runs are typically instantiated by passing the arguments to the parameters that are defined in pipelines. The arguments can be passed manually or within the trigger definition.

Parameters

Parameters are key-value pairs of read-only configuration. Parameters are defined in the pipeline. The arguments for the defined parameters are passed during execution from the run context that was created by a trigger or a pipeline that was executed manually. Activities within the pipeline consume the parameter values.

A dataset is a strongly typed parameter and a reusable/referenceable entity. An activity can reference datasets and can consume the properties that are defined in the dataset definition.

A linked service is also a strongly typed parameter that contains the connection information to either a data store or a compute environment. It is also a reusable/referenceable entity.

Control flow

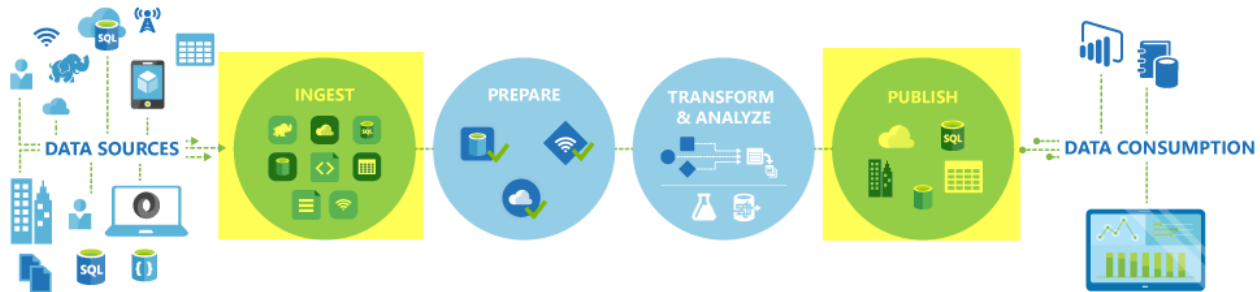
Control flow is an orchestration of pipeline activities that includes chaining activities in a sequence, branching, defining parameters at the pipeline level, and passing arguments while invoking the pipeline on-demand or from a trigger. It also includes custom-state passing and looping containers, that is, For-each iterators.

Variables

Variables can be used inside of pipelines to store temporary values and can also be used in conjunction with parameters to enable passing values between pipelines, data flows, and other activities.

Data Movement using ADF

In Azure Data Factory and Synapse pipelines, you can use the Copy activity to copy data among data stores located on-premises and in the cloud. After you copy the data, you can use other activities to further transform and analyze it. You can also use the Copy activity to publish transformation and analysis results for business intelligence (BI) and application consumption.



The Copy activity is executed on an integration runtime. You can use different types of integration runtimes for different data copy scenarios:

When you're copying data between two data stores that are publicly accessible through the internet from any IP, you can use the Azure integration runtime for the copy activity. This integration runtime is secure, reliable, scalable, and globally available.

When you're copying data to and from data stores that are located on-premises or in a network with access control (for example, an Azure virtual network), you need to set up a self-hosted integration runtime.

An integration runtime needs to be associated with each source and sink data store. For information about how the Copy activity determines which integration runtime to use, see [Determining which IR to use](#).

To copy data from a source to a sink, the service that runs the Copy activity performs these steps:

1. Reads data from a source data store.
2. Performs serialization/deserialization, compression/decompression, column mapping, and so on. It performs these operations based on the configuration of the input dataset, output dataset, and Copy activity.
3. Writes data to the sink/destination data store.

Supported file formats

Azure Data Factory supports the following file formats. Refer to each article for format-based settings.

- Avro format
- Binary format
- Delimited text format
- Excel format
- JSON format
- ORC format
- Parquet format
- XML format

You can use the Copy activity to copy files as-is between two file-based data stores, in which case the data is copied efficiently without any serialization or deserialization. In addition, you can also parse or generate files of a given format, for example, you can perform the following:

- Copy data from a SQL Server database and write to Azure Data Lake Storage Gen2 in Parquet format.
- Copy files in text (CSV) format from an on-premises file system and write to Azure Blob storage in Avro format.
- Copy zipped files from an on-premises file system, decompress them on-the-fly, and write extracted files to Azure Data Lake Storage Gen2.
- Copy data in Gzip compressed-text (CSV) format from Azure Blob storage and write it to Azure SQL Database.
- Many more activities that require serialization/deserialization or compression/decompression.

Supported regions

The service that enables the Copy activity is available globally in the regions and geographies listed in Azure integration runtime locations. The globally available topology ensures efficient data movement that usually avoids cross-region hops. See Products by region to check the availability of Data Factory, Synapse Workspaces and data movement in a specific region.

Read More: <https://docs.microsoft.com/en-us/azure/data-factory/copy-activity-overview>

Use the Copy Data tool in the Azure Data Factory Studio to copy data.

Azure roles

To create Data Factory instances, the user account that you use to sign in to Azure must be a member of the *contributor* or *owner* role, or an *administrator* of the Azure subscription. To view the permissions that you have in the subscription, go to the [Azure portal](#), select your username in the upper-right corner, select "..." icon for more options, and then select **My permissions**. If you have access to multiple subscriptions, select the appropriate subscription.

To create and manage child resources for Data Factory - including datasets, linked services, pipelines, triggers, and integration runtimes - the following requirements are applicable:

- To create and manage child resources in the Azure portal, you must belong to the **Data Factory Contributor** role at the resource group level or above.
- To create and manage child resources with PowerShell or the SDK, the **contributor** role at the resource level or above is sufficient.

Azure Storage account

You use a general-purpose Azure Storage account (specifically Blob storage) as both *source* and *destination* data stores in this quickstart. If you don't have a general-purpose Azure Storage account, see [Create a storage account](#) to create one.

Get the storage account name

You need the name of your Azure Storage account for this quickstart. The following procedure provides steps to get the name of your storage account:

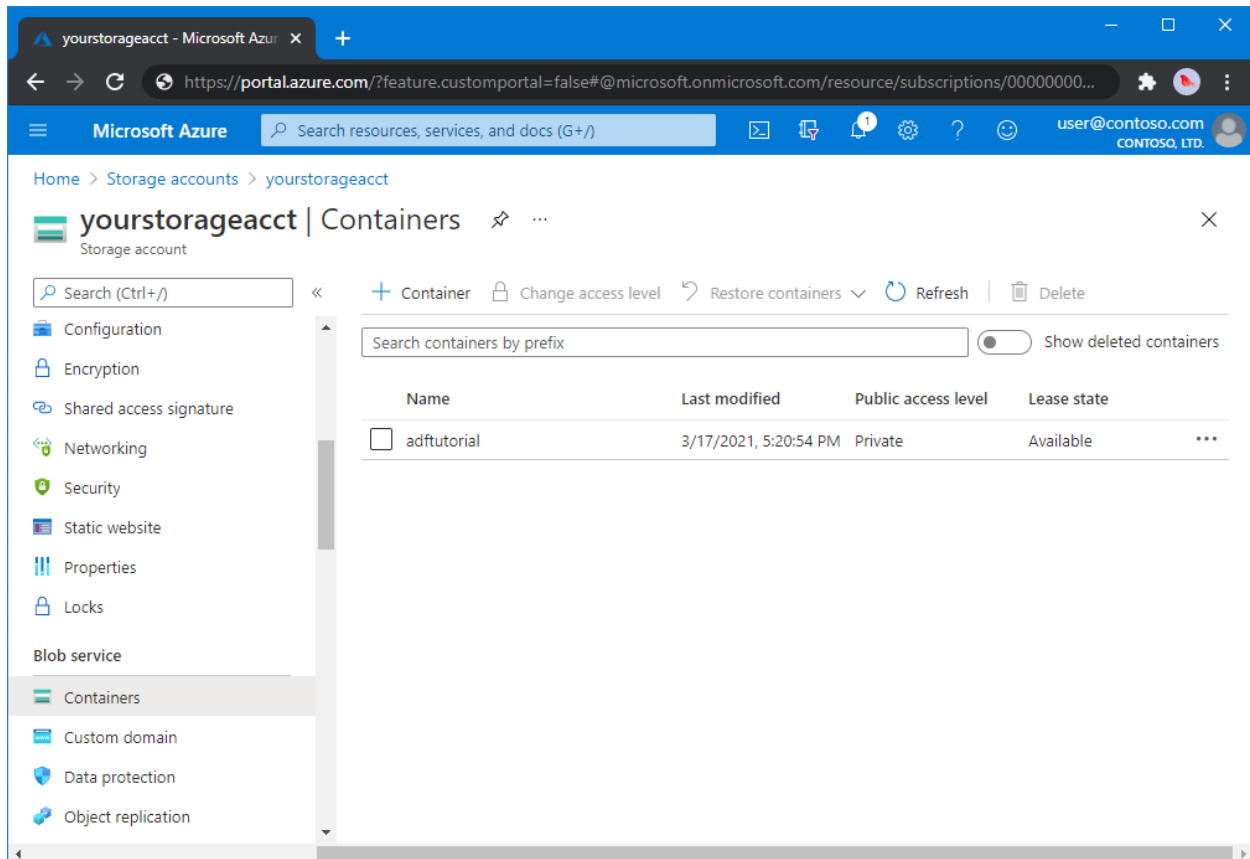
1. In a web browser, go to the [Azure portal](#) and sign in using your Azure username and password.
2. From the Azure portal menu, select **All services**, then select **Storage > Storage accounts**. You can also search for and select *Storage accounts* from any page.
3. In the **Storage accounts** page, filter for your storage account (if needed), and then select your storage account.

You can also search for and select *Storage accounts* from any page.

Create a blob container

In this section, you create a blob container named **adftutorial** in Azure Blob storage.

1. From the storage account page, select **Overview** > **Containers**.
2. On the <Account name> - **Containers** page's toolbar, select **Container**.
3. In the **New container** dialog box, enter **adftutorial** for the name, and then select **OK**.
The <Account name> - **Containers** page is updated to include **adftutorial** in the list of containers.



Add an input folder and file for the blob container

In this section, you create a folder named **input** in the container you created, and then upload a sample file to the input folder. Before you begin, open a text editor such as **Notepad**, and create a file named **emp.txt** with the following content:

emp.txtCopy

John, Doe
Jane, Doe

Save the file in the **C:\ADFv2QuickStartPSH** folder. (If the folder doesn't already exist, create it.) Then return to the Azure portal and follow these steps:

1. In the <Account name> - **Containers** page where you left off, select **adftutorial** from the updated list of containers.
1. If you closed the window or went to another page, sign in to the [Azure portal](#) again.
2. From the Azure portal menu, select **All services**, then select **Storage > Storage accounts**. You can also search for and select *Storage accounts* from any page.
3. Select your storage account, and then select **Containers > adftutorial**.
2. On the **adftutorial** container page's toolbar, select **Upload**.
3. In the **Upload blob** page, select the **Files** box, and then browse to and select the **emp.txt** file.
4. Expand the **Advanced** heading. The page now displays as shown:

Upload blob

adftutorial/

Files

"emp.txt"

☐ Overwrite if files already exist

Advanced

Authentication type

Azure AD user account **Account key**

Blob type

Block blob

☒ Upload .vhd files as page blobs (recommended)

Block size

4 MB

Access tier

Hot (Inferred)

Upload to folder

Blob index tags

Key	Value
<input type="text"/>	<input type="text"/>

Encryption scope

☒ Use existing default container scope

☐ Choose an existing scope

Retention policy

☒ No retention

☐ Choose custom retention period:

01/26/2022 4:54:38 PM

☐ Enable version-level immutability on the container to set a retention policy.

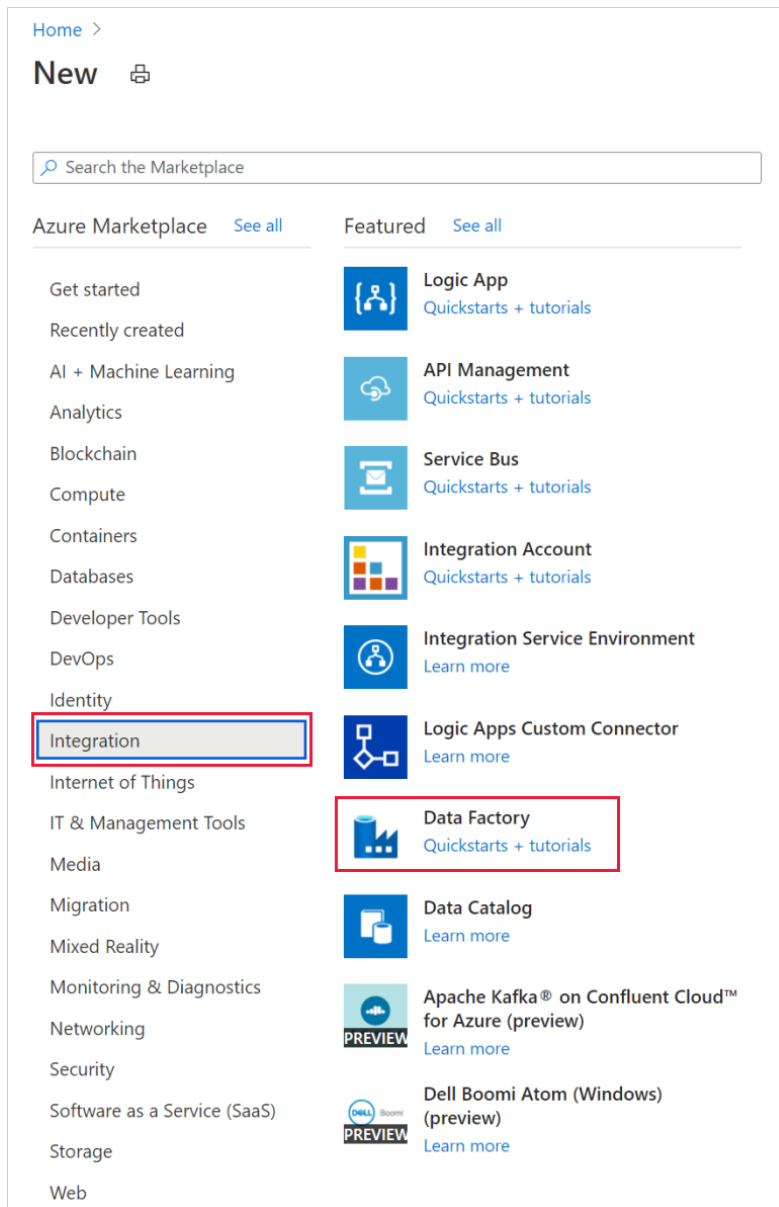
Upload

5. In the **Upload to folder** box, enter **input**.
6. Select the **Upload** button. You should see the **emp.txt** file and the status of the upload in the list.
7. Select the **Close** icon (an **X**) to close the **Upload blob** page

Keep the **adftutorial** container page open. You use it to verify the output at the end of this quickstart.

Create a data factory

1. Launch **Microsoft Edge** or **Google Chrome** web browser. Currently, Data Factory UI is supported only in Microsoft Edge and Google Chrome web browsers.
2. Go to the [Azure portal](#).
3. From the Azure portal menu, select **Create a resource** > **Integration** > **Data Factory**:



4. On the **New data factory** page, enter **ADFTutorialDataFactory** for **Name**.

The name of the Azure Data Factory must be *globally unique*. If you see the following error, change the name of the data factory (for example, **<yourname>ADFTutorialDataFactory**) and try creating again. For naming rules for Data Factory artifacts, see the [Data Factory - naming rules](#) article.

Create Data Factory ...

Basics Git configuration Networking Advanced Tags Review + create

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ	<your Azure subscription selection> ▼
Resource group * ⓘ	YourResourceGroup ▼
	Create new

Instance details

Region * ⓘ	South Central US ▼
Name * ⓘ	ADFTutorialDataFactory
	✖ The Data Factory name is already taken. Choose a different name.
Version * ⓘ	V2 ▼

- For **Subscription**, select your Azure subscription in which you want to create the data factory.
- For **Resource Group**, use one of the following steps:
 - Select **Use existing**, and select an existing resource group from the list.
 - Select **Create new**, and enter the name of a resource group.

To learn about resource groups, see [Using resource groups to manage your Azure resources](#).

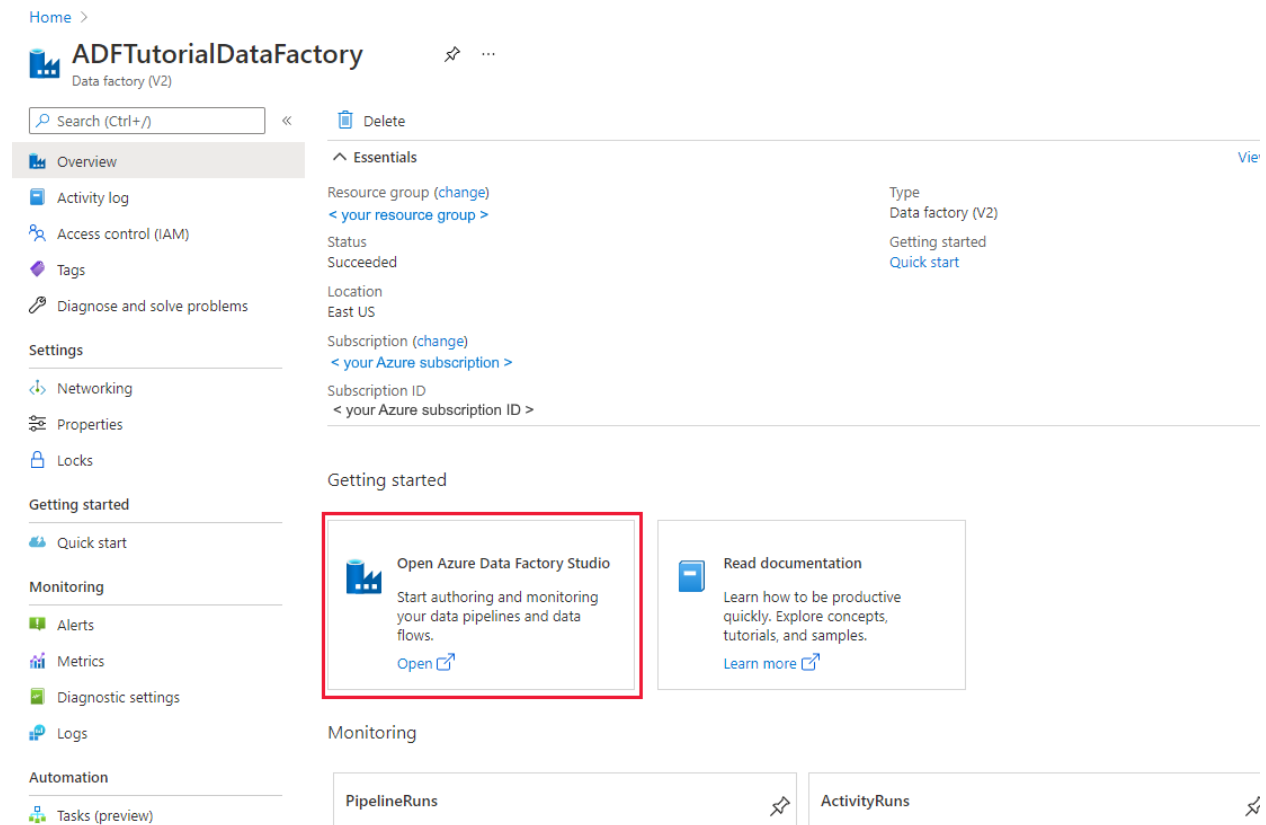
- For **Version**, select **V2**.
- For **Location**, select the location for the data factory.

The list shows only locations that Data Factory supports, and where your Azure Data Factory meta data will be stored. The associated data stores (like Azure Storage and Azure

SQL Database) and computes (like Azure HDInsight) that Data Factory uses can run in other regions.

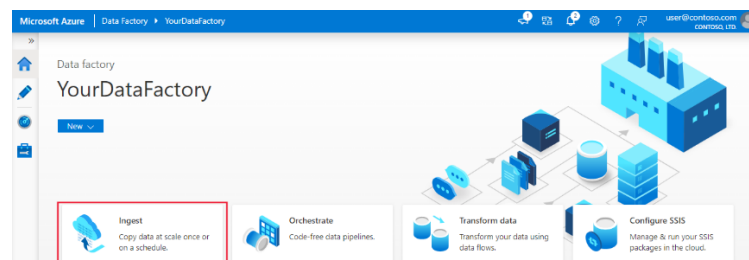
9. Select **Create**.

10. After the creation is complete, you see the **Data Factory** page. Select **Open** on the **Open Azure Data Factory Studio** tile to start the Azure Data Factory user interface (UI) application on a separate tab.



Start the Copy Data tool

1. On the home page of Azure Data Factory, select the **Ingest** tile to start the Copy Data tool.



2. On the **Properties** page of the Copy Data tool, choose **Built-in copy task** under **Task type**, then select **Next**.

Copy Data tool

1 Properties

2 Source

3 Target

4 Settings


5 Review and finish


Use Copy Data Tool to perform a one-time or scheduled data load from 90+ data sources. Follow the wizard experience to specify your data loading settings, and let the Copy Data Tool generate the artifacts for you, including pipelines, datasets, and linked services. [Learn more](#)

Properties

Select copy data task type and configure task schedule

Task type

 **Built-in copy task**
You will get single pipeline to copy data from 90+ data source easily.

 **Metadata-driven copy task (Preview)**
Metadata is required to be stored in external control tables to load data at large-scale.

You will get single pipeline to quickly copy objects from data source store to destination in a very intuitive manner.

Task cadence or task schedule *

☒ Run once now ☐ Schedule ☐ Tumbling window

< Previous **Next >** Cancel

3. On the **Source data store** page, complete the following steps:
 1. Click + **Create new connection** to add a connection.
 2. Select the linked service type that you want to create for the source connection. In this tutorial, we use **Azure Blob Storage**. Select it from the gallery, and then select **Continue**.
 3. On the New connection (Azure Blob Storage) page, specify a name for your connection. Select your Azure subscription from the Azure subscription list and your storage account from the Storage account name list, test connection, and then select Create.
 4. Select the newly created connection in the Connection block.
 5. In the File or folder section, select Browse to navigate to the adftutorial/input folder, select the emp.txt file, and then click OK.
 6. Select the Binary copy checkbox to copy file as-is, and then select Next.

New connection

 Search

All

Azure

Database

File

Generic protocol

NoSQL

Services and apps



Amazon Marketplace Web Service



Amazon Redshift



Amazon S3



Amazon S3 Compatible



Apache Impala



Azure Blob Storage



Azure Cosmos DB
(MongoDB API)



Azure Cosmos DB (SQL
API)



Azure Data Explorer
(Kusto)



Continue

Cancel

New connection (Azure Blob Storage)

Name *

AzureBlobStorage

Description

Connect via integration runtime * ⓘ

AutoResolveIntegrationRuntime

Authentication method

Account key

Connection string

Azure Key Vault

Account selection method ⓘ

☒ From Azure subscription ☐ Enter manually

Azure subscription ⓘ

Storage account name *



Additional connection properties

+ New

Test connection ⓘ

☒ To linked service ☐ To file path

Annotations

+ New

▸ Parameters

▸ Advanced ⓘ

Create

Back



Connection successful



Test connection

Cancel

Source data store

Specify the source data store for the copy task. You can use an existing data store connection or specify a new data store.

Source type

Connection * [Edit](#) [+ Create new connection](#)

File or folder *
If the identity you use to access the data store only has permission to subdirectory instead of the entire account, specify the path to browse.

[Browse](#)

Options

☒ Binary copy ⓘ

Compression type

☒ Recursively ⓘ

☐ Delete files after completion ⓘ

Max concurrent connections ⓘ

Filter by last modified

Start time (UTC) End time (UTC) ⓘ

< Previous

Next >

4. On the **Destination data store** page, complete the following steps:
 1. Select the **AzureBlobStorage** connection that you created in the **Connection** block.
 2. In the **Folder path** section, enter **adftutorial/output** for the folder path.

Destination data store

Specify the destination data store for the copy task. You can use an existing data store connection or specify a new data store.

Target type

Connection * [Edit](#) [+ Create new connection](#)

Folder path *

If the identity you use to access the data store only has permission to subdirectory instead of the entire account, specify the path to browse.

[Browse](#)

File name

Compression type

Copy behavior ⓘ

Max concurrent connections ⓘ

Block size (MB) ⓘ

[< Previous](#)[Next >](#)

3. Leave other settings as default and then select **Next**.
5. On the **Settings** page, specify a name for the pipeline and its description, then select **Next** to use other default configurations.

Settings

Enter name and description for the copy data task, more options for data movement

Task name *

Task description

Data consistency verification ☐

Enable logging ☐

Enable staging ☐

Advanced

Data integration unit

☐ Edit

You will be charged # of used DIUs * copy duration * \$0.25/DIU-hour. Local currency and separate discounting may apply per subscription type. [Learn more](#)

Degree of copy parallelism

☒ Edit



Preserve

[< Previous](#)


[Next >](#)




[Cancel](#)

6. On the **Summary** page, review all settings, and select **Next**.
7. On the **Deployment complete** page, select **Monitor** to monitor the pipeline that you created.

 Azure Blob Storage →  Azure Blob Storage

Deployment complete

▸ Validate copy runtime environment 





Deployment step	Status
> Creating datasets	Succeeded 
> Creating pipelines	Succeeded 
> Running pipelines	Succeeded 

Datasets and pipelines have been created. You can now monitor and edit the copy pipelines or click finish to close Copy Data Tool.

[Finish](#) [Edit pipeline](#) [Monitor](#)

8. The application switches to the **Monitor** tab. You see the status of the pipeline on this tab. Select **Refresh** to refresh the list. Click the link under **Pipeline name** to view activity run details or rerun the pipeline.


Pipeline runs

Triggered Debug  Rerun  Cancel ▾  Refresh  Edit columns [List](#) [Gantt](#)

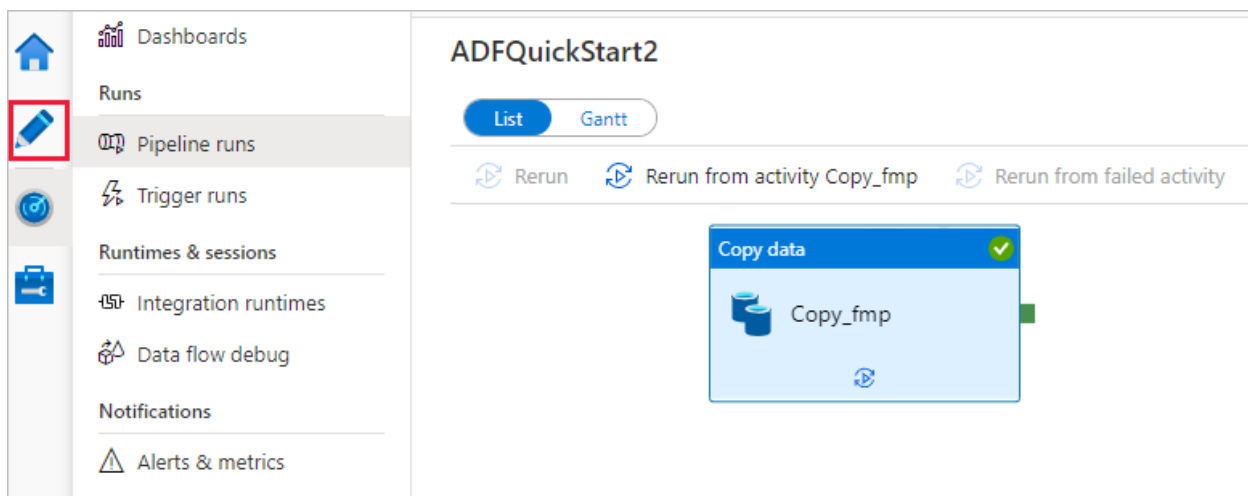
[Beijing, Chongqing, ...](#) : Last 24 hours Pipeline name : ADFQuickStart Status : All Runs : Latest runs

[Add filter](#)

Showing 1 - 1 items

<input type="checkbox"/> Pipeline name	Run start ↑↓	Run end	Duration	Triggered by	Status	Error
<input type="checkbox"/> ADFQuickStart	5/28/21, 4:21:56 PM	5/28/21, 4:22:04 PM	00:00:08	Manual trigger	 Succeeded	

9. On the Activity runs page, select the **Details** link (eyeglasses icon) under the **Activity name** column for more details about copy operation. For details about the properties, see [Copy Activity overview](#).
10. To go back to the Pipeline Runs view, select the **All pipeline runs** link in the breadcrumb menu. To refresh the view, select **Refresh**.
11. Verify that the **emp.txt** file is created in the **output** folder of the **adftutorial** container. If the output folder doesn't exist, the Data Factory service automatically creates it.
12. Switch to the **Author** tab above the **Monitor** tab on the left panel so that you can edit linked services, datasets, and pipelines. To learn about editing them in the Data Factory UI, see [Create a data factory by using the Azure portal](#).



Create a data factory by using the Azure portal and Azure Data Factory Studio

Azure Storage account

You use a general-purpose Azure Storage account (specifically Blob storage) as both *source* and *destination* data stores in this quickstart. If you don't have a general-purpose Azure Storage account, see [Create a storage account](#) to create one.

Get the storage account name

You need the name of your Azure Storage account for this quickstart. The following procedure provides steps to get the name of your storage account:

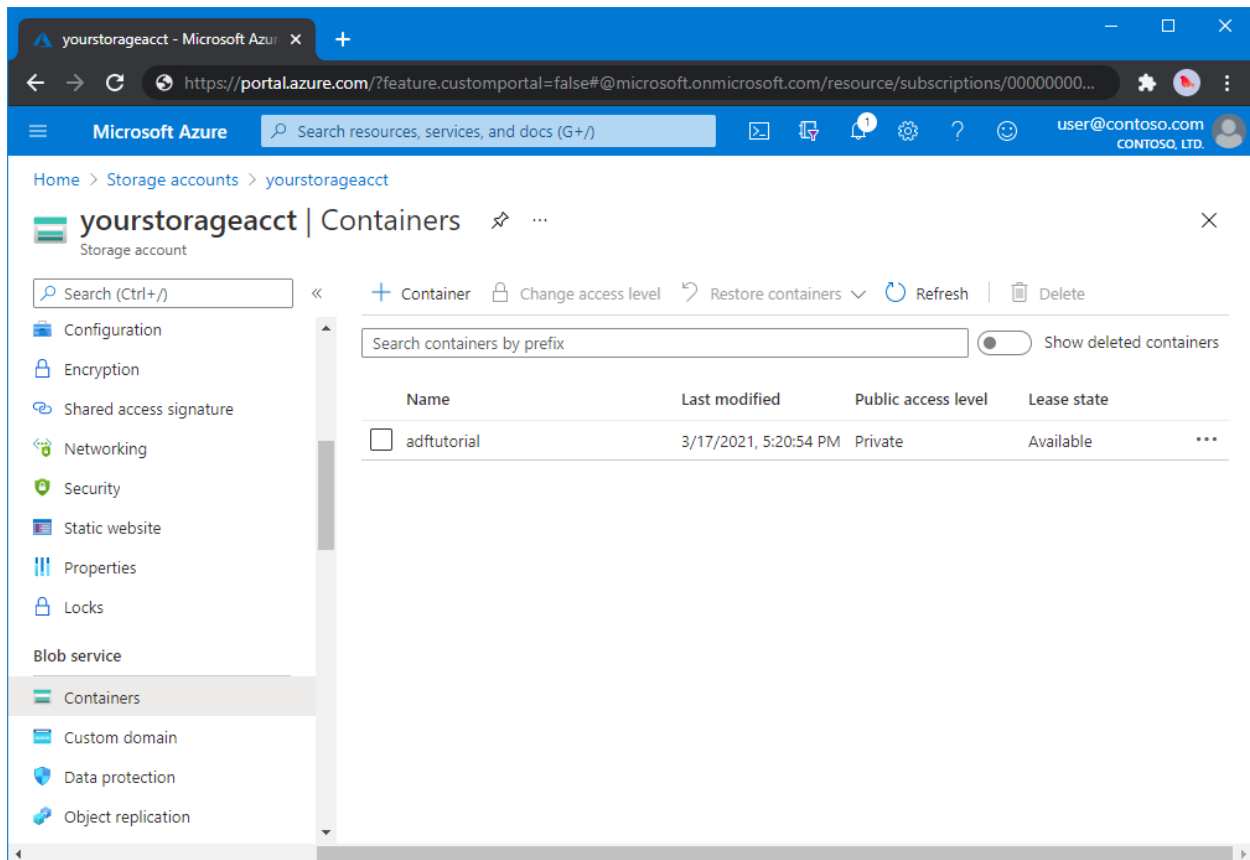
1. In a web browser, go to the [Azure portal](#) and sign in using your Azure username and password.
2. From the Azure portal menu, select **All services**, then select **Storage > Storage accounts**. You can also search for and select *Storage accounts* from any page.
3. In the **Storage accounts** page, filter for your storage account (if needed), and then select your storage account.

You can also search for and select *Storage accounts* from any page.

Create a blob container

In this section, you create a blob container named **adftutorial** in Azure Blob storage.

1. From the storage account page, select **Overview > Containers**.
2. On the <Account name> - **Containers** page's toolbar, select **Container**.
3. In the **New container** dialog box, enter **adftutorial** for the name, and then select **OK**. The <Account name> - **Containers** page is updated to include **adftutorial** in the list of containers.



Add an input folder and file for the blob container

In this section, you create a folder named **input** in the container you created, and then upload a sample file to the input folder. Before you begin, open a text editor such as **Notepad**, and create a file named **emp.txt** with the following content:

emp.txtCopy

John, Doe
Jane, Doe

Save the file in the **C:\ADfv2QuickStartPSH** folder. (If the folder doesn't already exist, create it.) Then return to the Azure portal and follow these steps:

1. In the *<Account name>* - **Containers** page where you left off, select **adftutorial** from the updated list of containers.
 1. If you closed the window or went to another page, sign in to the [Azure portal](#) again.
 2. From the Azure portal menu, select **All services**, then select **Storage** > **Storage accounts**. You can also search for and select *Storage accounts* from any page.
 3. Select your storage account, and then select **Containers** > **adftutorial**.
2. On the **adftutorial** container page's toolbar, select **Upload**.


3. In the **Upload blob** page, select the **Files** box, and then browse to and select the **emp.txt** file.
4. Expand the **Advanced** heading. The page now displays as shown:

Upload blob

✕

adftutorial/

Files ⓘ

"emp.txt" 

☐ Overwrite if files already exist

^ Advanced

Authentication type ⓘ

Azure AD user account **Account key**

Blob type ⓘ

Block blob ▾

☒ Upload .vhd files as page blobs (recommended)

Block size ⓘ

4 MB ▾

Access tier ⓘ

Hot (Inferred) ▾

Upload to folder

Blob index tags ⓘ

Key	Value

Encryption scope


☒ Use existing default container scope


☐ Choose an existing scope

Retention policy

☐ No retention

☐ Choose custom retention period:

01/26/2022  4:54:38 PM

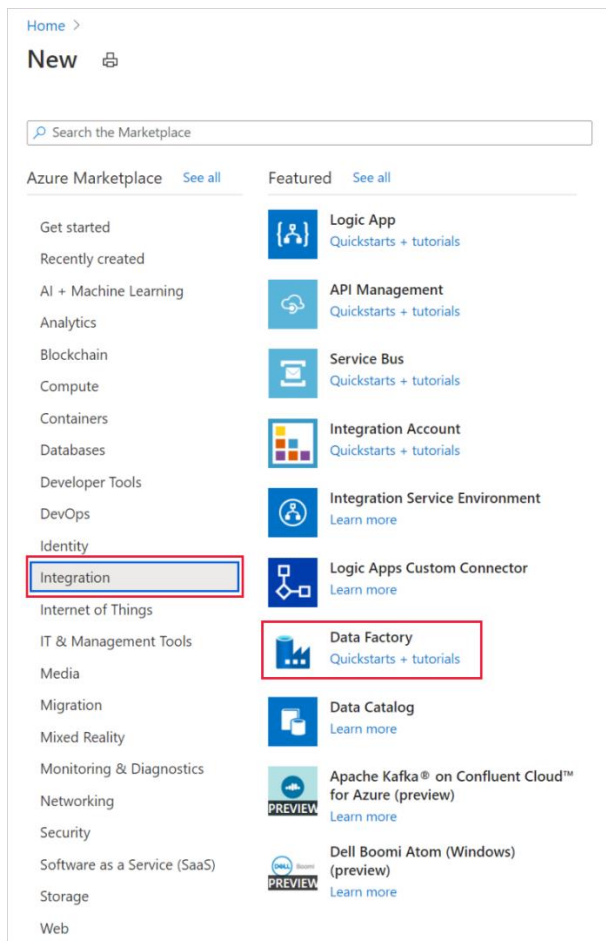
 Enable version-level immutability on the container to set a retention policy.

Upload

5. In the **Upload to folder** box, enter **input**.
6. Select the **Upload** button. You should see the **emp.txt** file and the status of the upload in the list.
7. Select the **Close** icon (an **X**) to close the **Upload blob** page.

Create a data factory

1. Launch **Microsoft Edge** or **Google Chrome** web browser. Currently, Data Factory UI is supported only in Microsoft Edge and Google Chrome web browsers.
2. Go to the [Azure portal](#).
3. From the Azure portal menu, select **Create a resource**.
4. Select **Integration**, and then select **Data Factory**.



5. On the **Create Data Factory** page, under **Basics** tab, select your Azure **Subscription** in which you want to create the data factory.
6. For **Resource Group**, take one of the following steps:

- a. Select an existing resource group from the drop-down list.
 - b. Select **Create new**, and enter the name of a new resource group.
7. For **Region**, select the location for the data factory.

The list shows only locations that Data Factory supports, and where your Azure Data Factory meta data will be stored. The associated data stores (like Azure Storage and Azure SQL Database) and computes (like Azure HDInsight) that Data Factory uses can run in other regions.

8. For **Name**, enter **ADFTutorialDataFactory**. The name of the Azure data factory must be *globally unique*. If you see the following error, change the name of the data factory (for example, **<yourname>ADFTutorialDataFactory**) and try creating again. For naming rules for Data Factory artifacts, see the [Data Factory - naming rules](#) article.

Create Data Factory ...

Basics Git configuration Networking Advanced Tags Review + create

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

<your Azure subscription selection>



Resource group * ⓘ

YourResourceGroup

[Create new](#)

Instance details

Region * ⓘ

South Central US

Name * ⓘ

ADFTutorialDataFactory

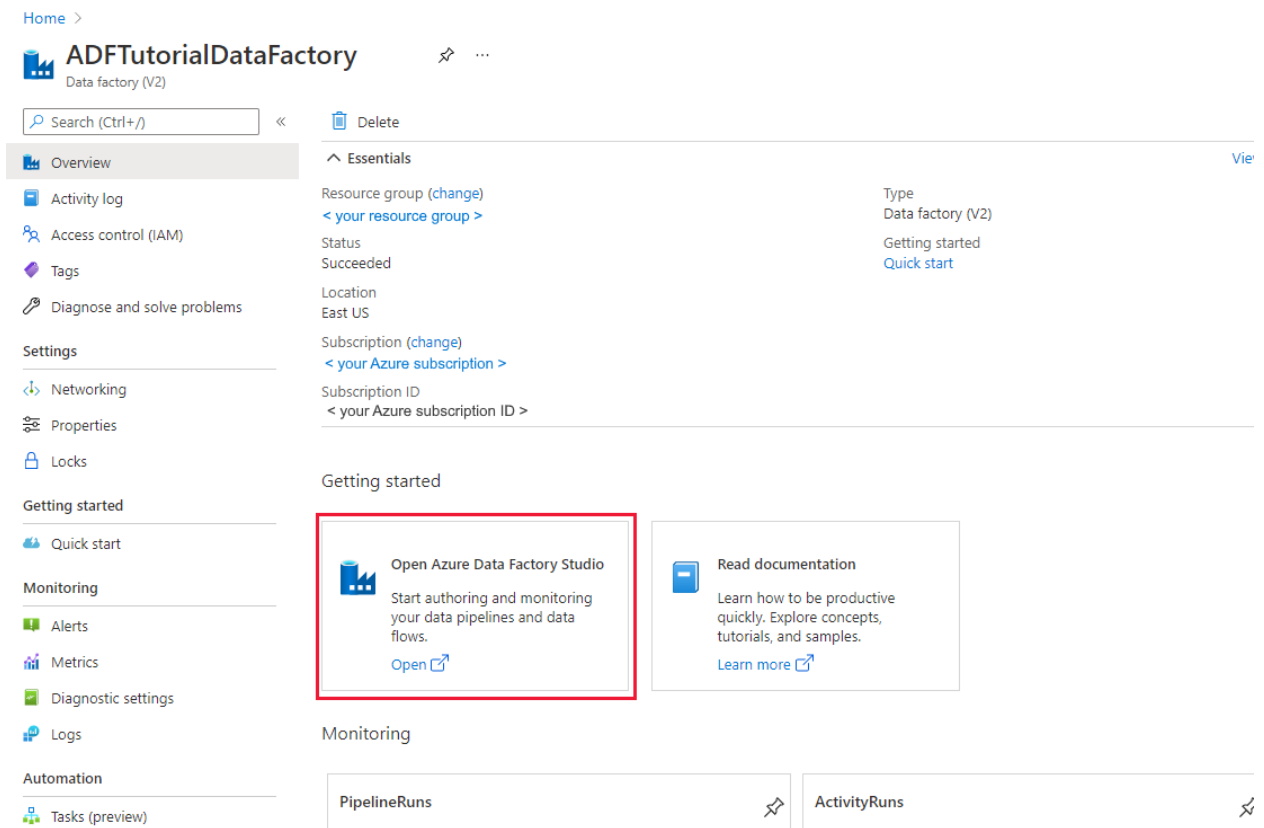
✖ The Data Factory name is already taken. Choose a different name.

Version * ⓘ

V2

9. For **Version**, select **V2**.
10. Select **Next: Git configuration**, and then select **Configure Git later** check box.
11. Select **Review + create**, and select **Create** after the validation is passed. After the creation is complete, select **Go to resource** to navigate to the **Data Factory** page.

12. Select **Open** on the **Open Azure Data Factory Studio** tile to start the Azure Data Factory user interface (UI) application on a separate browser tab.

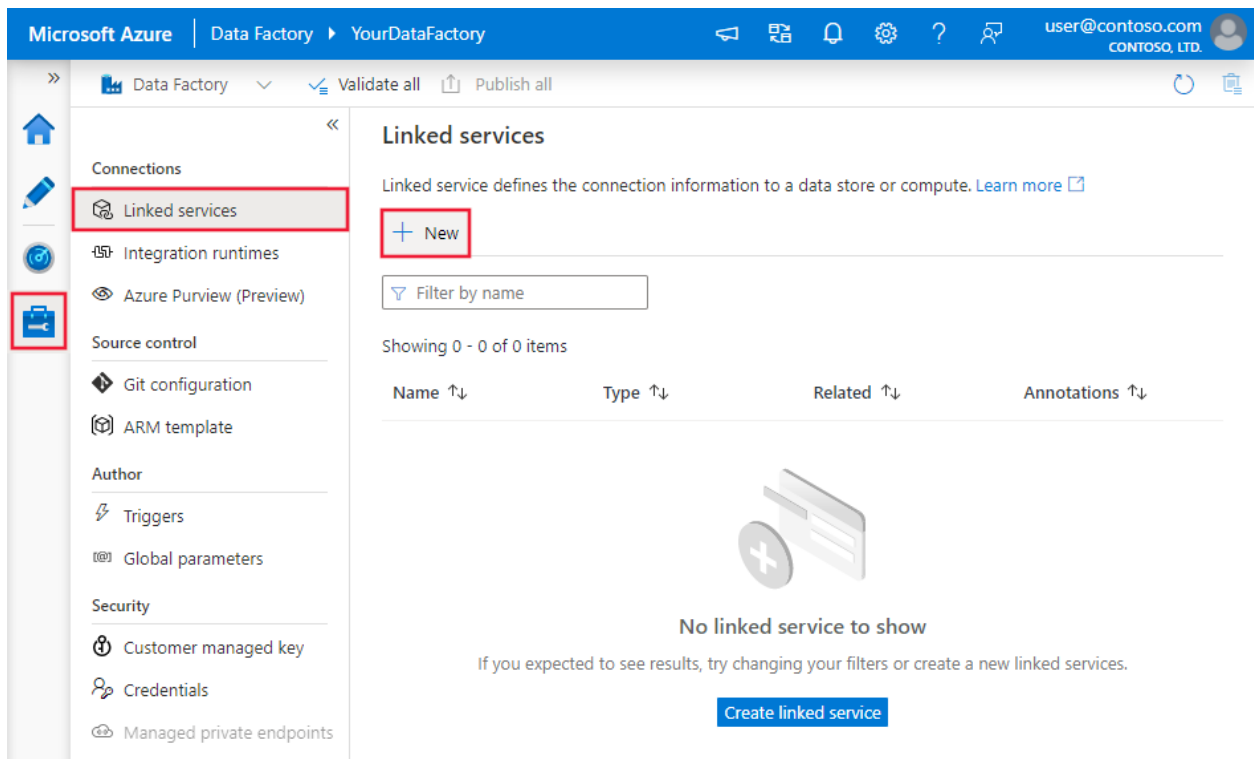


Note: If you see that the web browser is stuck at "Authorizing", clear the **Block third-party cookies and site data** check box. Or keep it selected, create an exception for **login.microsoftonline.com**, and then try to open the app again.

Create a linked service

In this procedure, you create a linked service to link your Azure Storage account to the data factory. The linked service has the connection information that the Data Factory service uses at runtime to connect to it.

1. On the Azure Data Factory UI page, open **Manage** tab from the left pane.
2. On the Linked services page, select **+New** to create a new linked service.



3. On the **New Linked Service** page, select **Azure Blob Storage**, and then select **Continue**.
4. On the New Linked Service (Azure Blob Storage) page, complete the following steps:
 - a. For **Name**, enter **AzureStorageLinkedService**.
 - b. For **Storage account name**, select the name of your Azure Storage account.
 - c. Select **Test connection** to confirm that the Data Factory service can connect to the storage account.
 - d. Select **Create** to save the linked service.

New linked service (Azure Blob Storage)

Name *

AzureStorageLinkedService

Description

Connect via integration runtime * ⓘ

AutoResolveIntegrationRuntime

Authentication method

Account key

Connection string

Azure Key Vault

Account selection method ⓘ

☒ From Azure subscription ☐ Enter manually

Azure subscription ⓘ

<select your Azure subscription here>

Storage account name *

<select your Storage account name here>

Additional connection properties

+ New

Test connection ⓘ

☒ To linked service ☐ To file path

Annotations

+ New

▸ Parameters

▸ Advanced ⓘ

Create

Back



Test connection

Cancel

Create datasets

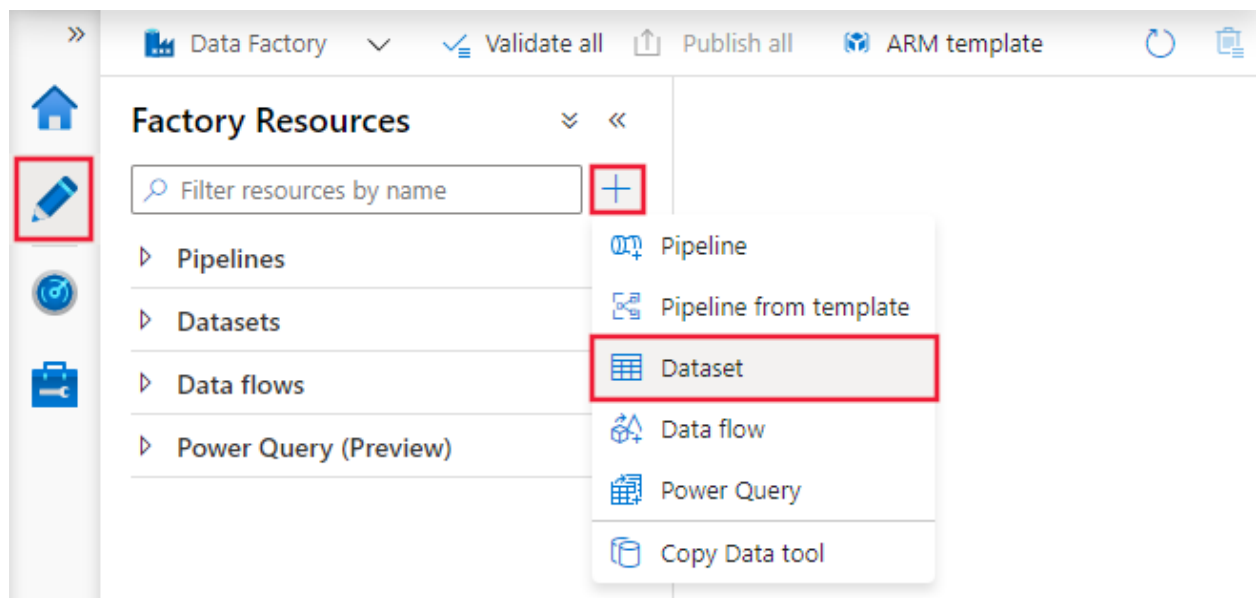
In this procedure, you create two datasets: **InputDataset** and **OutputDataset**. These datasets are of type **AzureBlob**. They refer to the Azure Storage linked service that you created in the previous section.

The input dataset represents the source data in the input folder. In the input dataset definition, you specify the blob container (**adftutorial**), the folder (**input**), and the file (**emp.txt**) that contain the source data.

The output dataset represents the data that's copied to the destination. In the output dataset definition, you specify the blob container (**adftutorial**), the folder (**output**), and the file to which the data is copied. Each run of a pipeline has a unique ID associated with it. You can access this ID by using the system variable **RunId**. The name of the output file is dynamically evaluated based on the run ID of the pipeline.

In the linked service settings, you specified the Azure Storage account that contains the source data. In the source dataset settings, you specify where exactly the source data resides (blob container, folder, and file). In the sink dataset settings, you specify where the data is copied to (blob container, folder, and file).

1. Select **Author** tab from the left pane.
2. Select the + (plus) button, and then select **Dataset**.











3. On the **New Dataset** page, select **Azure Blob Storage**, and then select **Continue**.

4. On the **Select Format** page, choose the format type of your data, and then select **Continue**. In this case, select **Binary** when copy files as-is without parsing the content.

Select format

Choose the format type of your data

 Avro	 Binary	 DelimitedText
 Excel	 Json	 ORC
 Parquet	 XML	

Continue

Back


Cancel

5. On the **Set Properties** page, complete following steps:
 - a. Under **Name**, enter **InputDataset**.

- b. For **Linked service**, select **AzureStorageLinkedService**.
- c. For **File path**, select the **Browse** button.
- d. In the **Choose a file or folder** window, browse to the **input** folder in the **adftutorial** container, select the **emp.txt** file, and then select **OK**.
- e. Select **OK**.

Set properties

Name

Linked service *
 

File path
 / / 

- 6. Repeat the steps to create the output dataset:
 - a. Select the + (plus) button, and then select **Dataset**.
 - b. On the **New Dataset** page, select **Azure Blob Storage**, and then select **Continue**.
 - c. On the **Select Format** page, choose the format type of your data, and then select **Continue**.
 - d. On the **Set Properties** page, specify **OutputDataset** for the name. Select **AzureStorageLinkedService** as linked service.
 - e. Under **File path**, enter **adftutorial/output**. If the **output** folder doesn't exist, the copy activity creates it at runtime.
 - f. Select **OK**.

Set properties

Name

OutputDataset

Linked service *

AzureStorageLinkedService

File path

adftutorial

/

output

/

File



OK

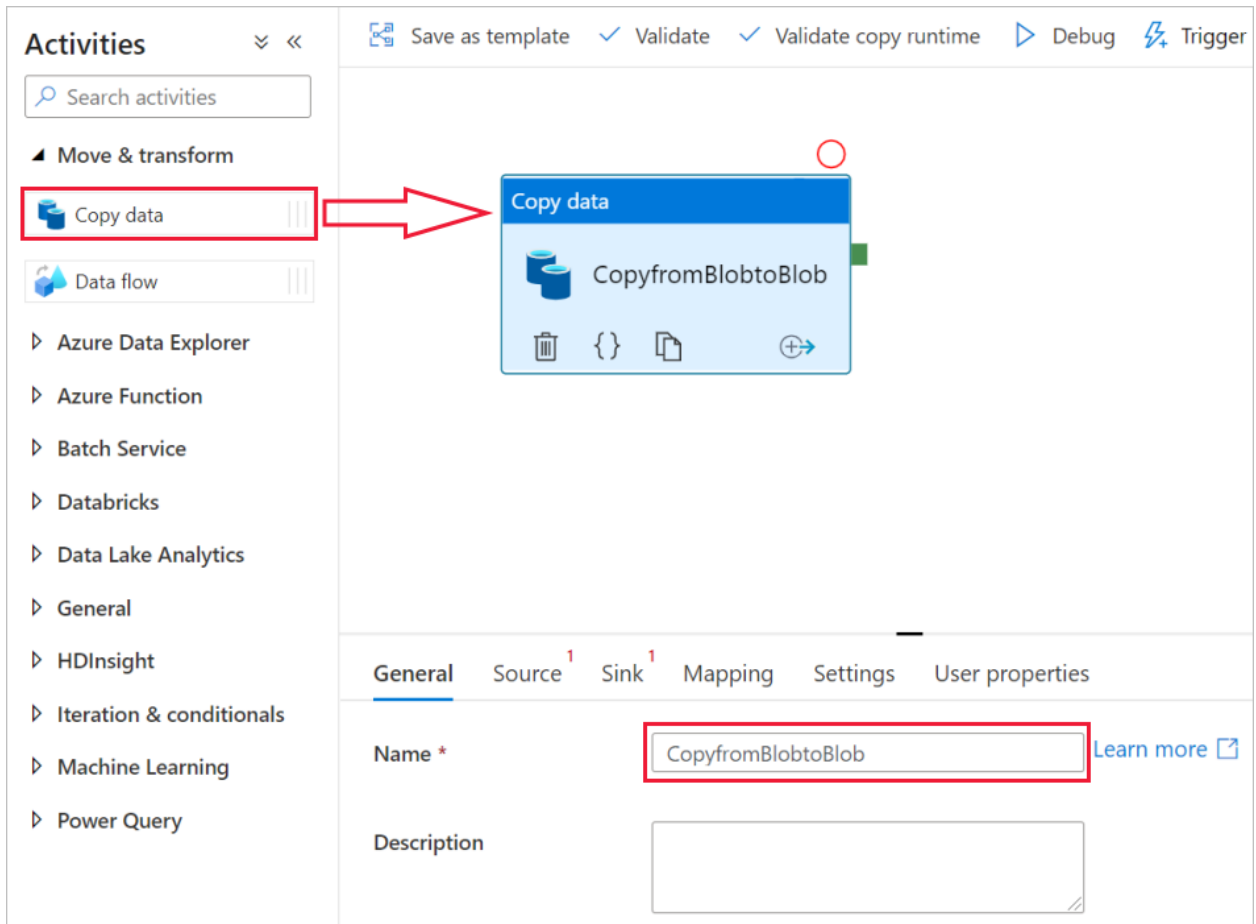
Back

Cancel

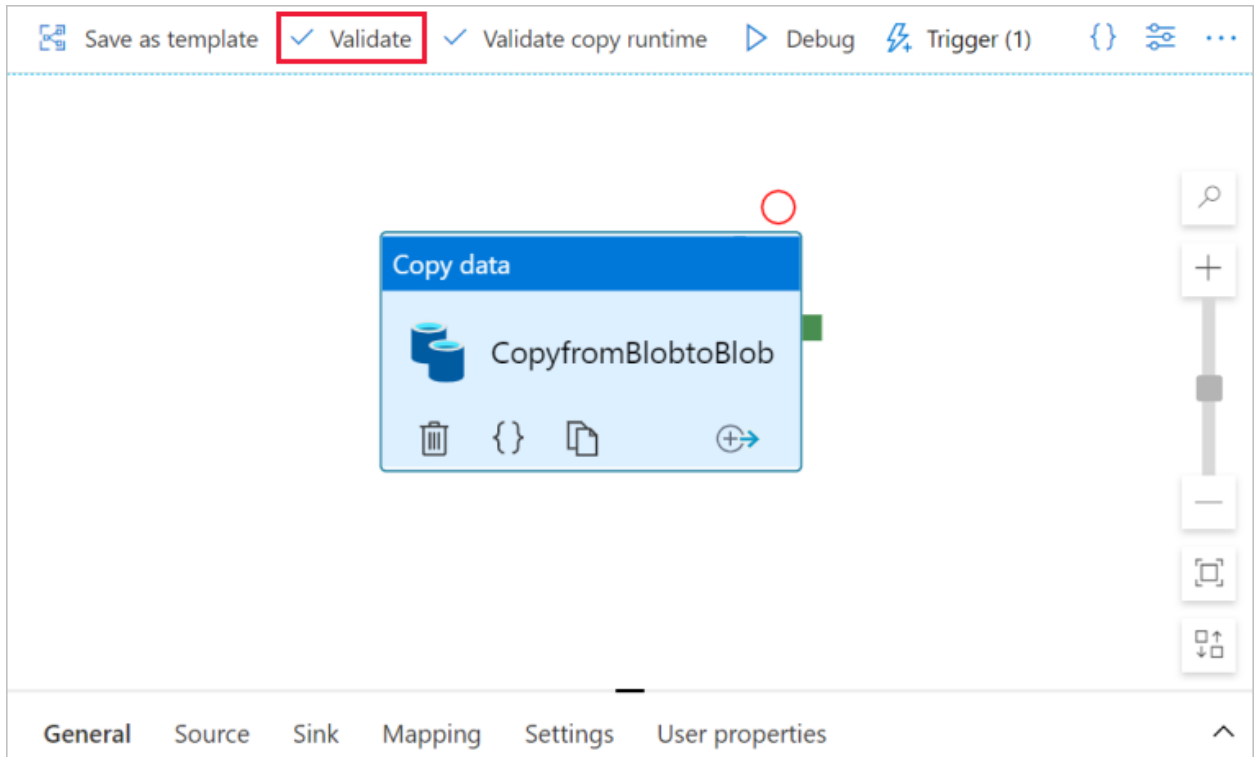
Create a pipeline

In this procedure, you create and validate a pipeline with a copy activity that uses the input and output datasets. The copy activity copies data from the file you specified in the input dataset settings to the file you specified in the output dataset settings. If the input dataset specifies only a folder (not the file name), the copy activity copies all the files in the source folder to the destination.

1. Select the + (plus) button, and then select **Pipeline**.
2. In the General panel under **Properties**, specify **CopyPipeline** for **Name**. Then collapse the panel by clicking the Properties icon in the top-right corner.
3. In the **Activities** toolbox, expand **Move & Transform**. Drag the **Copy Data** activity from the **Activities** toolbox to the pipeline designer surface. You can also search for activities in the **Activities** toolbox. Specify **CopyFromBlobToBlob** for **Name**.



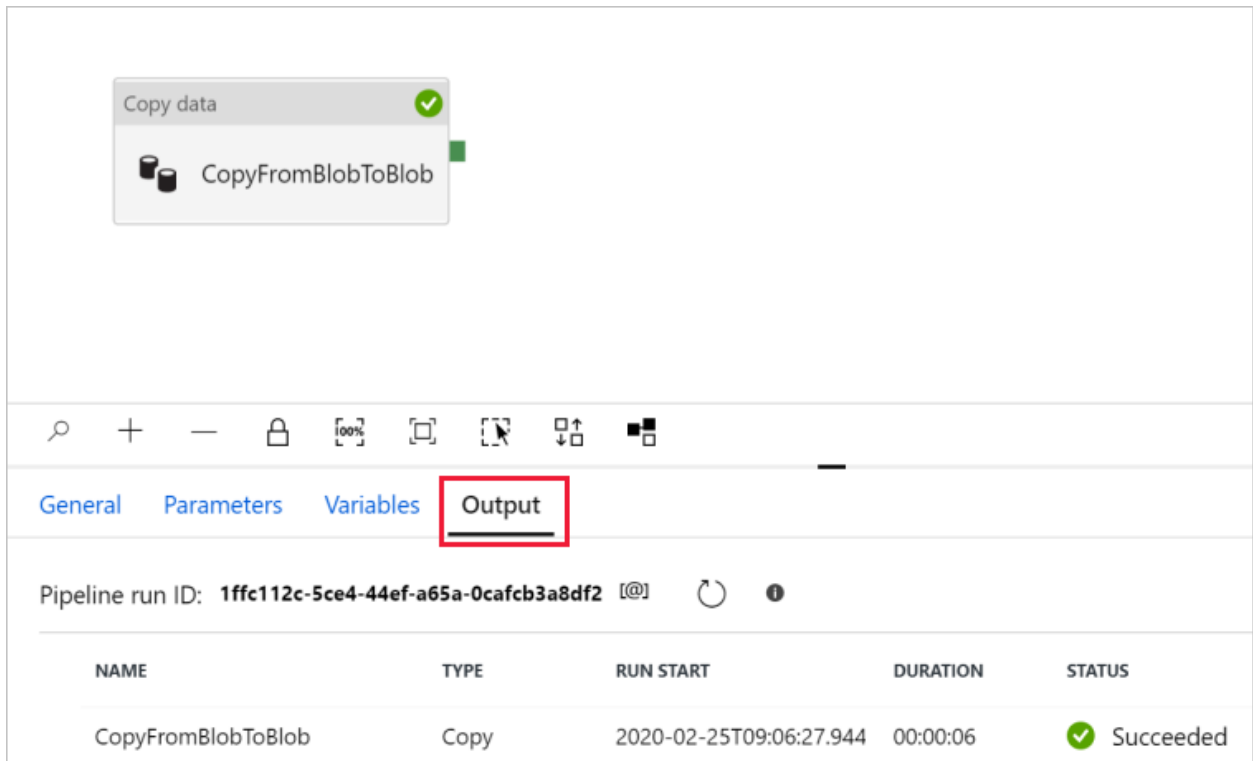
4. Switch to the **Source** tab in the copy activity settings, and select **InputDataset** for **Source Dataset**.
5. Switch to the **Sink** tab in the copy activity settings, and select **OutputDataset** for **Sink Dataset**.
6. Click **Validate** on the pipeline toolbar above the canvas to validate the pipeline settings. Confirm that the pipeline has been successfully validated. To close the validation output, select the Validation button in the top-right corner.



Debug the pipeline

In this step, you debug the pipeline before deploying it to Data Factory.

1. On the pipeline toolbar above the canvas, click **Debug** to trigger a test run.
2. Confirm that you see the status of the pipeline run on the **Output** tab of the pipeline settings at the bottom.

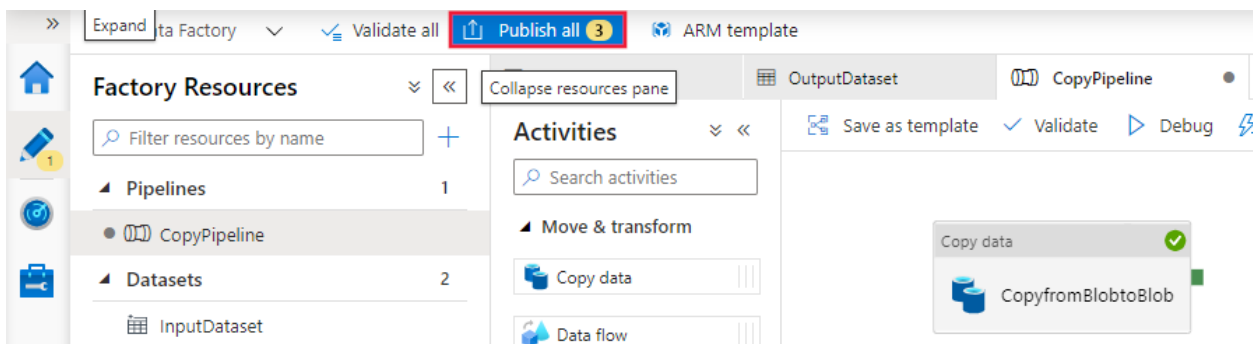


3. Confirm that you see an output file in the **output** folder of the **adftutorial** container. If the output folder doesn't exist, the Data Factory service automatically creates it.

Trigger the pipeline manually

In this procedure, you deploy entities (linked services, datasets, pipelines) to Azure Data Factory. Then, you manually trigger a pipeline run.

1. Before you trigger a pipeline, you must publish entities to Data Factory. To publish, select **Publish all** on the top.



2. To trigger the pipeline manually, select **Add Trigger** on the pipeline toolbar, and then select **Trigger Now**. On the **Pipeline run** page, select **OK**.

Monitor the pipeline

1. Switch to the **Monitor** tab on the left. Use the **Refresh** button to refresh the list.

Dashboard navigation: Dashboards, Runs, **Pipeline runs**, Trigger runs, Runtimes & sessions, Integration runtimes.

Pipeline runs interface: Triggered, Debug, Rerun, Cancel, Refresh, Edit columns, List, Gantt.

Search by run ID or name. Local time: Last 24 hours. Pipeline name: All. Status: All. Add filter.

Showing 1 - 3 items

<input type="checkbox"/> Pipeline name	Run start ↑↓	Duration	Status ↑↓	Triggered by	Error
<input type="checkbox"/> CopyPipeline	3/16/21, 9:02:54 PM	00:00:08	✓ Succeeded	Manual trigger	

2. Select the **CopyPipeline** link, you'll see the status of the copy activity run on this page.
3. To view details about the copy operation, select the **Details** (eyeglasses image) link. For details about the properties, see [Copy Activity overview](#).

All pipeline runs > CopyPipeline - Activity runs

CopyPipeline

List, Gantt

Refresh, Edit pipeline

Pipeline was modified after this run. The current pipeline configuration is shown.

Copy data

CopyfromBlobtoBlob

Activity runs

Pipeline run ID 773ed5dc-382d-4dfa-b5bf-0c5bac34ff4e

All status ▾

Showing 1 - 1 of 1 items

Activity name	Activity type	Run start ↑↓	Duration	Status	Integration runtime
CopyfromBlo...	Copy data	3/16/21, 9:02:57 PM	00:00:06	✓ Succeeded	DefaultIntegrationRuntime (South Central US)

4. Confirm that you see a new file in the **output** folder.
5. You can switch back to the **Pipeline runs** view from the **Activity runs** view by selecting the **All pipeline runs** link.

Trigger the pipeline on a schedule

This procedure is optional in this tutorial. You can create a *scheduler trigger* to schedule the pipeline to run periodically (hourly, daily, and so on). In this procedure, you create a trigger to run every minute until the end date and time that you specify.

1. Switch to the **Author** tab.
2. Go to your pipeline, select **Add Trigger** on the pipeline toolbar, and then select **New/Edit**.
3. On the **Add Triggers** page, select **Choose trigger**, and then select **New**.
4. On the **New Trigger** page, under **End**, select **On Date**, specify an end time a few minutes after the current time, and then select **OK**.

A cost is associated with each pipeline run, so specify the end time only minutes apart from the start time. Ensure that it's the same day. However, ensure that there's enough time for the pipeline to run between the publish time and the end time. The trigger comes into effect only after you publish the solution to Data Factory, not when you save the trigger in the UI.

5. On the **New Trigger** page, select the **Activated** check box, and then select **OK**.

New trigger

Name *
trigger1

Description

Type *
Schedule

Start date * ⓘ
03/17/2021 3:00 AM

Time zone * ⓘ
Coordinated Universal Time (UTC)

Recurrence * ⓘ
Every 15 Minute(s)

☒ Specify an end date

End On * ⓘ
03/17/2021 5:00 AM

Annotations
+ New

Activated * ⓘ
☒ Yes ☐ No

OK Cancel

6. Review the warning message, and select **OK**.
7. Select **Publish all** to publish changes to Data Factory.
8. Switch to the **Monitor** tab on the left. Select **Refresh** to refresh the list. You see that the pipeline runs once every minute from the publish time to the end time.

Notice the values in the **TRIGGERED BY** column. The manual trigger run was from the step (**Trigger Now**) that you did earlier.

9. Switch to the **Trigger runs** view.
10. Confirm that an output file is created for every pipeline run until the specified end date and time in the **output** folder.

Data Transformation using Mapping Data Flow

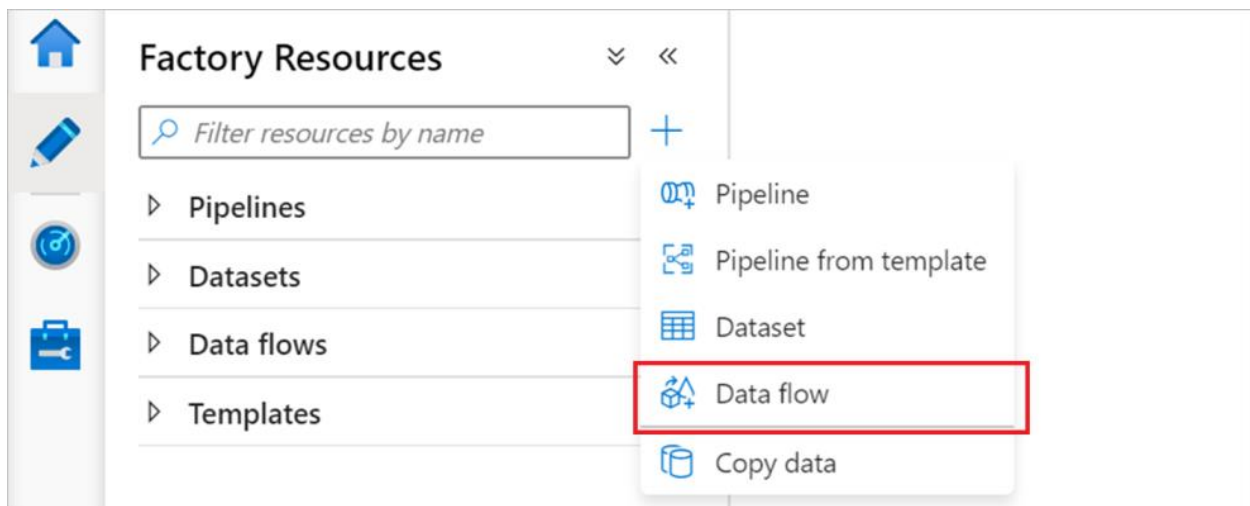
What are mapping data flows?

Mapping data flows are visually designed data transformations in Azure Data Factory. Data flows allow data engineers to develop data transformation logic without writing code. The resulting data flows are executed as activities within Azure Data Factory pipelines that use scaled-out Apache Spark clusters. Data flow activities can be operationalized using existing Azure Data Factory scheduling, control, flow, and monitoring capabilities.

Mapping data flows provide an entirely visual experience with no coding required. Your data flows run on ADF-managed execution clusters for scaled-out data processing. Azure Data Factory handles all the code translation, path optimization, and execution of your data flow jobs.

Getting started

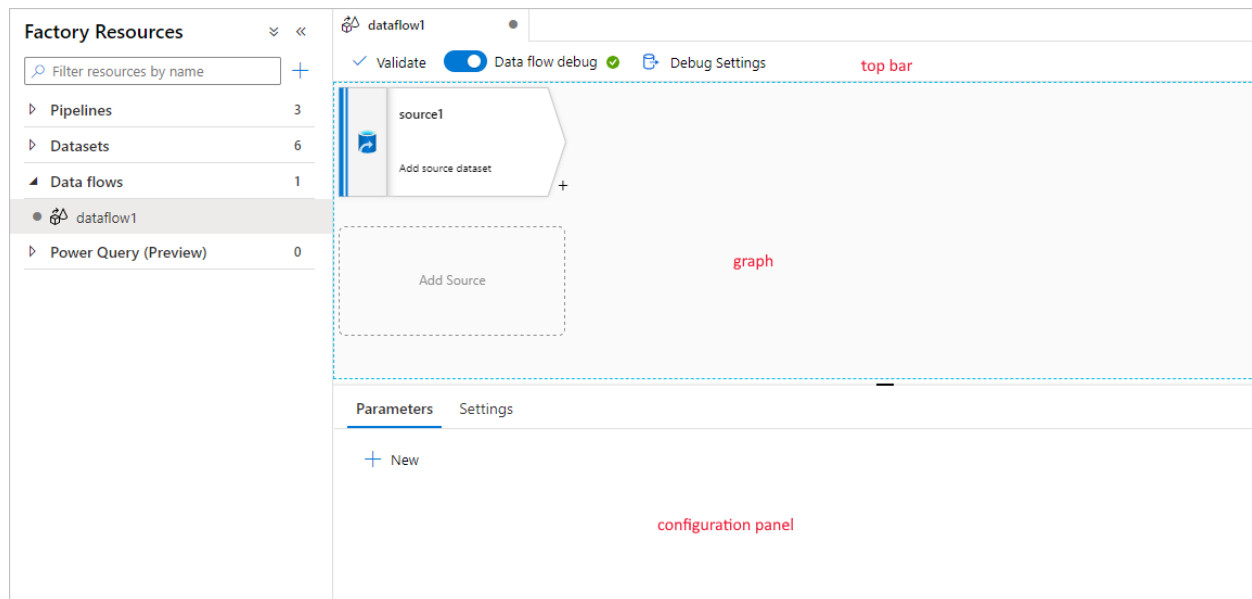
Data flows are created from the factory resources pane like pipelines and datasets. To create a data flow, select the plus sign next to **Factory Resources**, and then select **Data Flow**.



This action takes you to the data flow canvas, where you can create your transformation logic. Select **Add source** to start configuring your source transformation.

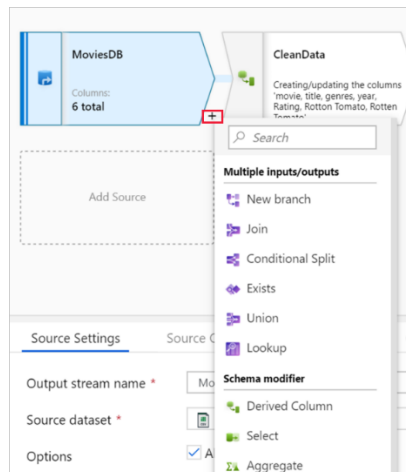
Authoring data flows

Mapping data flow has a unique authoring canvas designed to make building transformation logic easy. The data flow canvas is separated into three parts: the top bar, the graph, and the configuration panel.



Graph

The graph displays the transformation stream. It shows the lineage of source data as it flows into one or more sinks. To add a new source, select **Add source**. To add a new transformation, select the plus sign on the lower right of an existing transformation.



Configuration panel

The configuration panel shows the settings specific to the currently selected transformation. If no transformation is selected, it shows the data flow. In the overall data flow configuration, you can add parameters via the **Parameters** tab.

Each transformation contains at least four configuration tabs.

Transformation settings

The first tab in each transformation's configuration pane contains the settings specific to that transformation. For more information, see that transformation's documentation page.

The screenshot shows the 'Source settings' tab of a configuration panel. It includes the following elements:

- Source settings** (selected), **Source options**, **Projection**, **Optimize**, **Inspect**, **Data preview**
- Output stream name ***: Text input field with 'Source' and a [Learn more](#) link.
- Source type ***: Dropdown menu with 'Dataset' selected.
- Dataset ***: Dropdown menu with 'ADLSGen2Input' selected. To the right are buttons: [Test connection](#), [Open](#), and [+ New](#).
- Options**:
 - ☒ **Allow schema drift** ⓘ
 - ☐ **Infer drifted column types** ⓘ
 - ☐ **Validate schema** ⓘ
- Skip line count**: Text input field.
- Sampling ***: Radio buttons for ☐ **Enable** and ☒ **Disable** ⓘ.

Optimize

The **Optimize** tab contains settings to configure partitioning schemes. To learn more about how to optimize your data flows, see the [mapping data flow performance guide](#).

The screenshot shows the 'Optimize' tab of a configuration panel. It includes the following elements:

- Aggregate settings**, **Optimize** (selected), **Inspect**, **Data preview**
- Partition option ***: Radio buttons for ☐ **Use current partitioning**, ☐ **Single partition**, and ☒ **Set Partitioning**.
- Partition type ***: Five visual diagrams representing different partitioning schemes: Round Robin, Hash, Dynamic Range, Fixed Range, and Key. The Round Robin diagram is highlighted with a blue border.
- Number of partitions ***: Text input field with '20'.

Inspect

The **Inspect** tab provides a view into the metadata of the data stream that you're transforming. You can see column counts, the columns changed, the columns added, data types, the column order, and column references. **Inspect** is a read-only view of your metadata. You don't need to have debug mode enabled to see metadata in the **Inspect** pane.

Derived column's settings

Optimize

Inspect

Data Preview

Description

Output schema

Input schema

Number of columns

New 1

Updated 2

Unchanged 4

Total 7

Order	Column	Type	Updated	Based on
1	movie	abc string		
2	title	abc string	*	title
3	genres	abc string		
4	year	12l long	*	year
5	Rating	abc string		
6	Rotten Tomato	abc string		
7	Rotten Tomato	12l long	*	Rotten Tomato

As you change the shape of your data through transformations, you'll see the metadata changes flow in the **Inspect** pane. If there isn't a defined schema in your source transformation, then metadata won't be visible in the **Inspect** pane. Lack of metadata is common in schema drift scenarios.

Data preview

If debug mode is on, the **Data Preview** tab gives you an interactive snapshot of the data at each transform.

Top bar

The top bar contains actions that affect the whole data flow, like saving and validation. You can view the underlying JSON code and data flow script of your transformation logic as well.

Available transformations

View the [mapping data flow transformation overview](#) to get a list of available transformations.

Data flow data types

- array
- binary
- boolean
- complex
- decimal (includes precision)
- date
- float
- integer
- long
- map
- short
- string
- timestamp

Data flow activity

Mapping data flows are operationalized within ADF pipelines using the [data flow activity](#). All a user has to do is specify which integration runtime to use and pass in parameter values.

Debug mode

Debug mode allows you to interactively see the results of each transformation step while you build and debug your data flows. The debug session can be used both in when building your data flow logic and running pipeline debug runs with data flow activities.

Monitoring data flows

Mapping data flow integrates with existing Azure Data Factory monitoring capabilities. To learn how to understand data flow monitoring output, see [monitoring mapping data flows](#).

The Azure Data Factory team has created a [performance tuning guide](#) to help you optimize the execution time of your data flows after building your business logic.

Transforming data using Mapping Data Flow

Mapping Data Flows provide an environment for building a wide range of data transformations visually without the need to use code. The resulting data flows that are created are subsequently executed on scaled-out Apache Spark clusters that are automatically provisioned when you execute the Mapping Data Flow. Mapping Data Flows also provides the capability to monitor the execution of the transformations so that you can view how the transformations are progressing, or to understand any errors that may occur

Transforming data using compute resources

Azure Data Factory can also call on compute resources to transform data by a data platform service that may be better suited to the job. A great example of this is that Azure Data Factory can create a pipeline to an analytical data platform such as Spark pools in an Azure Synapse Analytics instance to perform a complex calculation using python. Another example could be to send data to an Azure SQL Database instance to execute a stored procedure using Transact-SQL. There is a wide range of compute resource, and the associated activities that they can perform as shown in the following table:

TRANSFORMING DATA USING COMPUTE RESOURCES	
Compute environment	activities
On-demand HDInsight cluster or your own HDInsight cluster	Hive, Pig, Spark, MapReduce, Hadoop Streaming
Azure Batch	Custom activities
Azure Machine Learning Studio Machine	Learning activities: Batch Execution and Update Resource
Azure Machine Learning	Azure Machine Learning Execute Pipeline
Azure Data Lake Analytics	Data Lake Analytics U-SQL
Azure SQL, Azure SQL Data Warehouse, SQL Server	Stored Procedure
Azure Databricks	Notebook, Jar, Python
Azure Function	Azure Function activity

Transforming data using SQL Server Integration Services (SSIS) packages

Many organizations have decades of development investment in SSIS packages that contain both ingestion and transformation logic from on-premises and cloud data stores. Azure Data Factory provides the ability to lift and shift existing SSIS workload, by creating an Azure-SSIS Integration Runtime to natively execute SSIS packages. Using Azure-SSIS Integration Runtime will enable you to deploy and manage your existing SSIS packages with little to no change using familiar tools such as SQL Server Data Tools (SSDT) and SQL Server Management Studio (SSMS), just like using SSIS on premises.

Describe Azure Data Factory transformation types

Mapping Data Flows provides a number of different transformations types that enable you to modify data. They are broken down into the following categories:

DESCRIBE AZURE DATA FACTORY TRANSFORMATION TYPES	
Category Name	Description
Schema modifier transformations	These types of transformations will make a modification to a sink destination by creating new columns based on the action of the transformation. An example of this is the Derived Column transformation that will create a new column based on the operations performed on existing column.
Row modifier transformations	These types of transformations impact how the rows are presented in the destination. An example of this is a Sort transformation that orders the data.
Multiple inputs/outputs transformations	These types of transformations will generate new data pipelines or merge pipelines into one. An example of this is the Union transformation that combines multiple data streams.

Below is a list of transformations that are available in the Mapping Data Flows

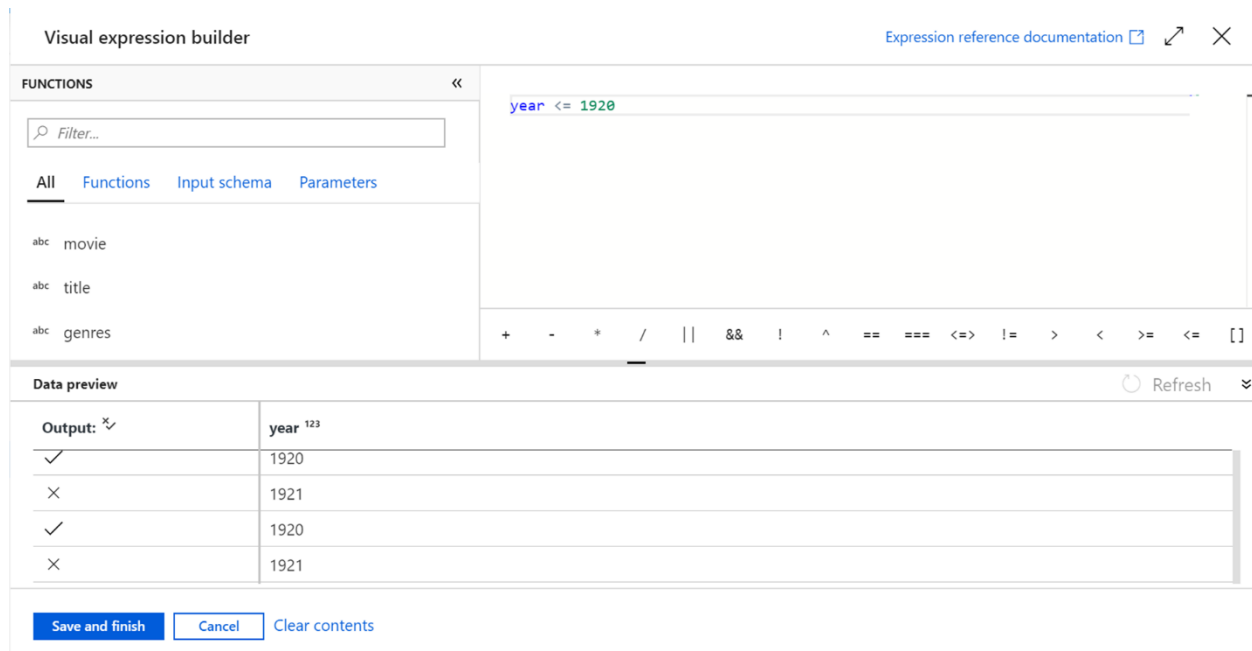
Name	Category	Description
Aggregate	Schema modifier	Define different types of aggregations such as SUM, MIN, MAX, and COUNT grouped by existing or computed columns.
Alter row	Row modifier	Set insert, delete, update, and upsert policies on rows. You can add one-to-many conditions as expressions. These conditions should be specified in order of priority, as each row will be marked with the policy corresponding to the first-matching expression. Each of those conditions can result in a row (or rows) being inserted, updated, deleted, or upserted. Alter Row can produce both DDL & DML actions against your database.
Conditional split	Multiple inputs/outputs	Route rows of data to different streams based on matching conditions.

Derived column	Schema modifier	generate new columns or modify existing fields using the data flow expression language.
Exists	Multiple inputs/outputs	Check whether your data exists in another source or stream.
Filter	Row modifier	Filter a row based upon a condition.
Flatten	Schema modifier	Take array values inside hierarchical structures such as JSON and unroll them into individual rows.
Join	Multiple inputs/outputs	Combine data from two sources or streams.
Lookup	Multiple inputs/outputs	Enables you to reference data from another source.
New branch	Multiple inputs/outputs	Apply multiple sets of operations and transformations against the same data stream.
Pivot	Schema modifier	An aggregation where one or more grouping columns has distinct row values transformed into individual columns.
Select	Schema modifier	Alias columns and stream names, and drop or reorder columns.
Sink	-	A final destination for your data.
Sort	Row modifier	Sort incoming rows on the current data stream.
Source	-	A data source for the data flow.
Surrogate key	Schema modifier	Add an incrementing non-business arbitrary key value.
Union	Multiple inputs/outputs	Combine multiple data streams vertically.
Unpivot	Schema modifier	Pivot columns into row values.
Window	Schema modifier	Define window-based aggregations of columns in your data streams.

Data Flow Expression Builder

Some of the transformations that you can define have an **Data Flow Expression Builder** that will enable you to customize the functionality of a transformation using columns, fields, variables, parameters, functions from your data flow in these boxes.

To build the expression, use the Expression Builder, which is launched by clicking in the expression text box inside the transformation. You'll also sometimes see "Computed Column" options when selecting columns for transformation. When you click that, you'll also see the Expression Builder launched.



The Expression Builder tool defaults to the text editor option. The auto-complete feature reads from the entire Azure Data Factory Data Flow object model with syntax checking and highlighting.

transforming data with the Mapping Data Flow

You can natively perform data transformations with Azure Data Factory code free using the Mapping Data Flow task. Mapping Data Flows provide a fully visual experience with no coding required. Your data flows will run on your own execution cluster for scaled-out data processing. Data flow activities can be operationalized via existing Data Factory scheduling, control, flow, and monitoring capabilities.

When building data flows, you can enable debug mode, which turns on a small interactive Spark cluster. Turn on debug mode by toggling the slider at the top of the authoring module. Debug clusters take a few minutes to warm up, but can be used to interactively preview the output of your transformation logic.

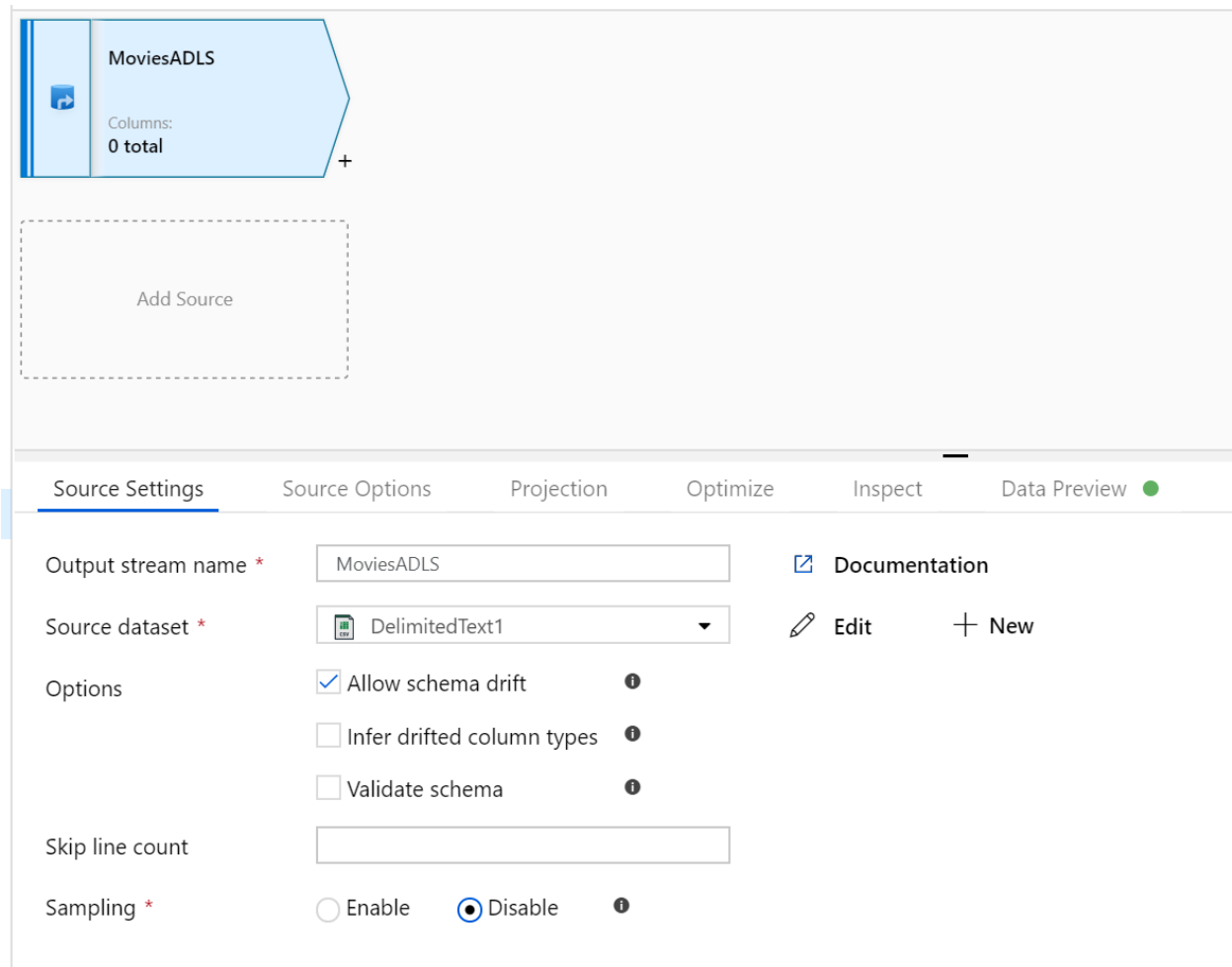


With the Mapping Data Flow added, and the Spark cluster running, this will enable you to perform the transformation, and run and preview the data. No coding is required as

Azure Data Factory handles all the code translation, path optimization, and execution of your data flow jobs.

Adding source data to the Mapping Data Flow

Open the Mapping Data Flow canvas. Click on the Add Source button in the Data Flow canvas. In the source dataset dropdown, select your data source, in this case the ADLS Gen2 dataset is used in this example



There are a couple of points to note:

- If your dataset is pointing at a folder with other files and you only want to use one file, you may need to create another dataset or utilize parameterization to make sure only a specific file is read

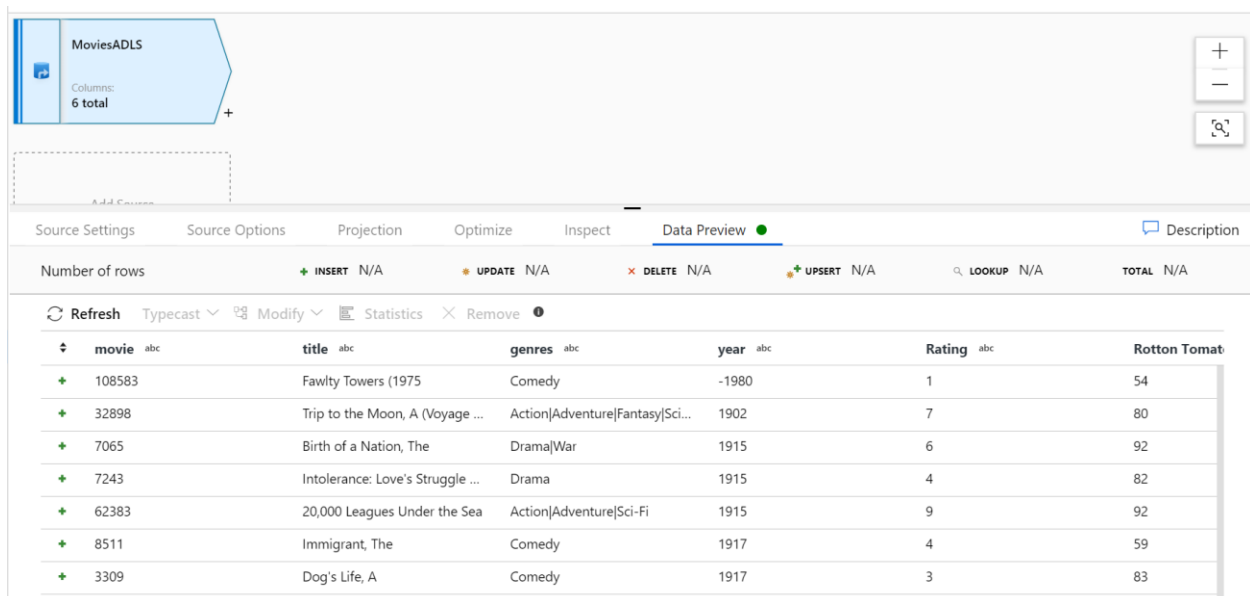
- If you have not imported your schema in your ADLS, but have already ingested your data, go to the dataset's 'Schema' tab and click 'Import schema' so that your data flow knows the schema projection.

Mapping Data Flow follows an extract, load, transform (ELT) approach and works with staging datasets that are all in Azure. Currently the following datasets can be used in a source transformation:

- Azure Blob Storage (JSON, Avro, Text, Parquet)
- Azure Data Lake Storage Gen1 (JSON, Avro, Text, Parquet)
- Azure Data Lake Storage Gen2 (JSON, Avro, Text, Parquet)
- Azure Synapse Analytics
- Azure SQL Database
- Azure CosmosDB

Azure Data Factory has access to over 80 native connectors. To include data from those other sources in your data flow, use the Copy Activity to load that data into one of the supported staging areas.

Once your debug cluster is warmed up, verify your data is loaded correctly via the Data Preview tab. Once you click the refresh button, Mapping Data Flow will show a snapshot of what your data looks like when it is at each transformation.



The screenshot shows the 'Data Preview' tab in the Mapping Data Flow interface. At the top, a blue box labeled 'MoviesADLS' indicates 'Columns: 6 total'. Below this, a table displays the data. The table has columns: 'movie', 'title', 'genres', 'year', 'Rating', and 'Rotton Tomat'. The data is sorted by 'movie' in descending order. The first row shows '108583' for movie, 'Fawltly Towers (1975)' for title, 'Comedy' for genres, '-1980' for year, '1' for Rating, and '54' for Rotton Tomat. The table is followed by a 'Refresh' button and a 'Typecast' dropdown menu. The 'Statistics' tab is also visible.

movie	title	genres	year	Rating	Rotton Tomat
108583	Fawltly Towers (1975)	Comedy	-1980	1	54
32898	Trip to the Moon, A (Voyage ...	Action Adventure Fantasy Sci...	1902	7	80
7065	Birth of a Nation, The	Drama War	1915	6	92
7243	Intolerance: Love's Struggle ...	Drama	1915	4	82
62383	20,000 Leagues Under the Sea	Action Adventure Sci-Fi	1915	9	92
8511	Immigrant, The	Comedy	1917	4	59
3309	Dog's Life, A	Comedy	1917	3	83

Using transformations in the Mapping Data Flow

Now that you have moved the data into Azure Data Lake Store Gen2, you are ready to build a Mapping Data Flow that will transform your data at scale via a spark cluster and then load it into a Data Warehouse.

The main tasks for this are as follows:

1. Preparing the environment
2. Adding a Data Source
3. Using Mapping Data Flow transformation
4. Writing to a Data Sink

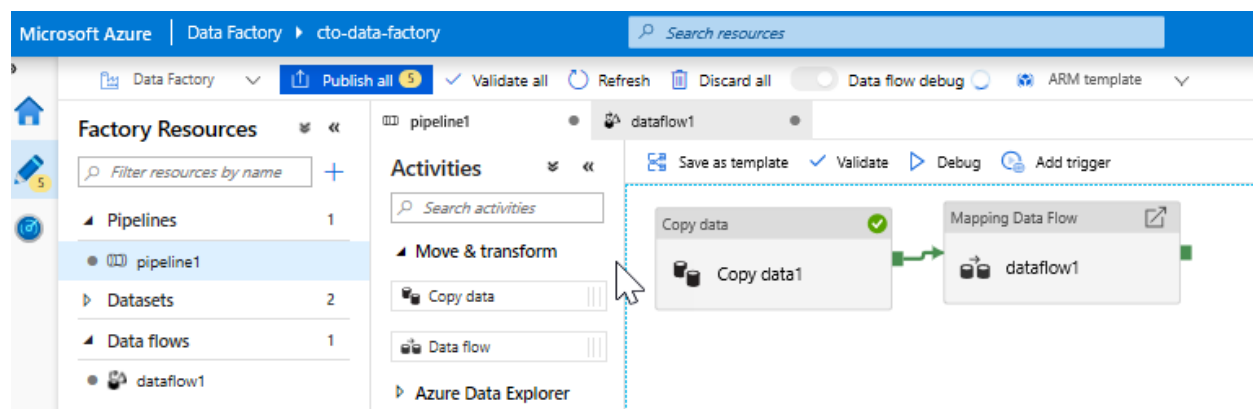
Task 1: Preparing the environment

1. **Turn on Data Flow Debug** Turn the **Data Flow Debug** slider located at the top of the authoring module on.

Note

Data Flow clusters take 5-7 minutes to warm up.

2. **Add a Data Flow activity.** In the Activities pane, open the Move and Transform accordion and drag the **Data Flow** activity onto the pipeline canvas. In the blade that pops up, click **Create new Data Flow** and select **Mapping Data Flow** and then click **OK**. Click on the **pipeline1** tab and drag the green box from your Copy activity to the Data Flow Activity to create an on success condition. You will see the following in the canvass:



Task 2: Adding a Data Source

1. **Add an ADLS source.** Double-click on the Mapping Data Flow object in the canvas. Click on the Add Source button in the Data Flow canvas. In the **Source dataset** dropdown, select your **ADLSG2** dataset used in your Copy activity

The screenshot shows the Azure Data Factory Mapping Data Flow interface. At the top, there is a canvas area with a blue arrow-shaped button labeled 'MoviesADLS' and 'Columns: 0 total'. Below this is a dashed box labeled 'Add Source'. The bottom section contains a tabbed interface with the following tabs: 'Source Settings' (selected), 'Source Options', 'Projection', 'Optimize', 'Inspect', and 'Data Preview' (with a green status indicator). The 'Source Settings' tab displays the following configuration:

- Output stream name *: MoviesADLS
- Source dataset *: DelimitedText1
- Options:
 - ☒ Allow schema drift
 - ☐ Infer drifted column types
 - ☐ Validate schema
- Skip line count: (empty text box)
- Sampling *: ☐ Enable ☒ Disable

On the right side of the 'Source Settings' tab, there are links for 'Documentation', 'Edit', and '+ New'.

- If your dataset is pointing at a folder with other files, you may need to create another dataset or utilize parameterization to make sure only the moviesDB.csv file is read
- If you have not imported your schema in your ADLS, but have already ingested your data, go to the dataset's 'Schema' tab and click 'Import schema' so that your data flow knows the schema projection.

Once your debug cluster is warmed up, verify your data is loaded correctly via the Data Preview tab. Once you click the refresh button, Mapping Data

Flow will show a snapshot of what your data looks like when it is at each transformation.

Task 3: Using Mapping Data Flow transformation

1. **Add a Select transformation to rename and drop a column.** In the preview of the data, you may have noticed that the "Rotten Tomatoes" column is misspelled. To correctly name it and drop the unused Rating column, you can add a [Select transformation](#) by clicking on the + icon next to your ADLS source node and choosing Select under Schema modifier.

Microsoft Azure | Data Factory | cto-data-factory

Search resources

Data Factory | Publish all | Validate all | Refresh | Discard all | Data flow debug | ARM template

Factory Resources

Filter resources by name

- Pipelines 1
 - pipeline1
- Datasets 2
- Data flows 1
 - dataflow1

pipeline1 | dataflow1

Validate | Debug Settings

source1
Columns: 6 total

Add Source

- Multiple inputs/outputs
 - Conditional Split
 - Exists
- Schema modifier
 - Select
 - Surrogate Key
- Row modifier
 - Sort
- Destination
 - Sink

Source settings | Source options | Projection | Optimize | Inspect | Data preview

Output stream name * source1 [Learn more](#)

Source dataset * ADLSG2 [Open](#) [New](#)

Options

- ☒ Allow schema drift
- ☐ Infer drifted column types
- ☐ Validate schema
- ☐ Multiline rows

Skip line count

Sampling * ☐ Enable ☒ Disable

In the **Name** as field, change 'Rotton' to 'Rotten'. To drop the Rating column, hover over it and click on the trash can icon.

Select settings Optimize Inspect Data preview ● Description ^

Output stream name * Select1 [Learn more](#)

Incoming stream * source1

Options

- ☒ Skip duplicate input columns ⓘ
- ☒ Skip duplicate output columns ⓘ

Input columns * ☐ Auto mapping ⓘ [Reset](#) [+ Add mapping](#) [Delete](#) 6 mappings: All inputs mapped

	source1's column		Name as	
:: <input type="checkbox"/>	abc movie	→	movie	+ Delete
:: <input type="checkbox"/>	abc title	→	title	+ Delete
:: <input type="checkbox"/>	abc genres	→	genres	+ Delete
:: <input type="checkbox"/>	abc year	→	year	+ Delete
:: <input type="checkbox"/>	abc Rating	→	Rating	+ Delete
:: <input type="checkbox"/>	abc Rotten Tomato	→	Rotten Tomato	+ Delete

2. **Add a Filter Transformation to filter out unwanted years.** Say you are only interested in movies made after 1951. You can add a [Filter transformation](#) to specify a filter condition by clicking on the **+** icon next to your Select transformation and choosing **Filter** under Row Modifier. Click on the **expression box** to open up the [Expression builder](#) and enter in your filter condition. Using the syntax of the [Mapping Data Flow expression language](#), `toInteger(year) > 1950` will convert the string year value to an integer and filter rows if that value is above 1950.

Filter settings Optimize Inspect Data preview ●

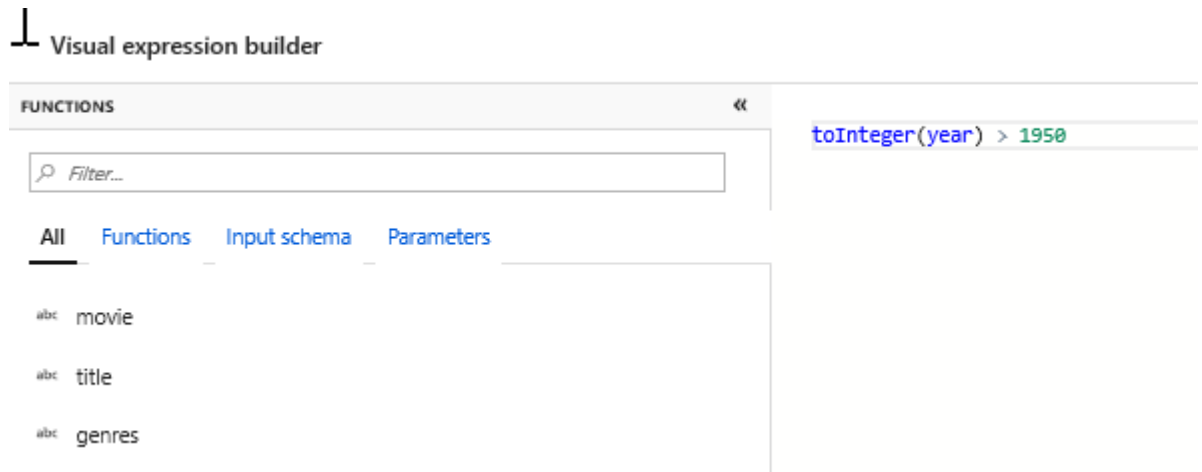
Output stream name * Filter1 [Learn more](#)

Incoming stream * Select1

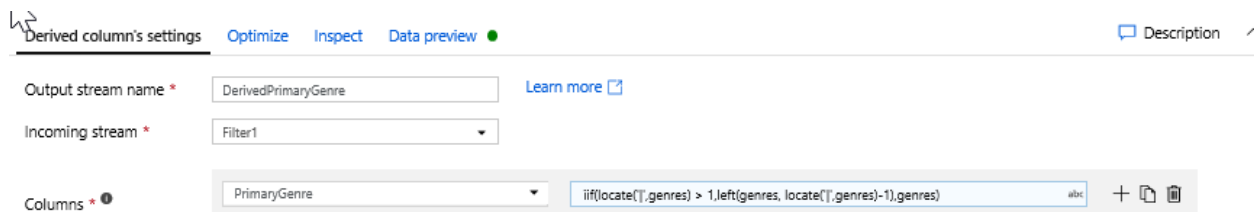
Filter on *

Enter filter... AND

You can use the expression builder's embedded Data preview pane to verify your condition is working properly



3. **Add a Derive Transformation to calculate primary genre.** As you may have noticed, the genres column is a string delimited by a '|' character. If you only care about the *first* genre in each column, you can derive a new column named **PrimaryGenre** via the [Derived Column](#) transformation by clicking on the **+** icon next to your Filter transformation and choosing Derived under Schema Modifier. Similar to the filter transformation, the derived column uses the Mapping Data Flow expression builder to specify the values of the new column.



In this scenario, you are trying to extract the first genre from the genres column, which is formatted as 'genre1|genre2|...|genreN'. Use the **locate** function to get the first 1-based index of the '|' in the genres string. Using the **iif** function, if this index is greater than 1, the primary genre can be calculated via the **left** function, which returns all characters in a string to the left of an index. Otherwise, the PrimaryGenre value is equal to the genres field. You can verify the output via the expression builder's Data preview pane.

4. **Rank movies via a Window Transformation.** Say you are interested in how a movie ranks within its year for its specific genre. You can add a [Window transformation](#) to define window-based aggregations by clicking on the **+** icon next to your Derived Column transformation and clicking

Window under Schema modifier. To accomplish this, specify what you are windowing over, what you are sorting by, what the range is, and how to calculate your new window columns. In this example, we will window over PrimaryGenre and year with an unbounded range, sort by Rotten Tomato descending, and calculate a new column called RatingsRank that is equal to the rank each movie has within its specific genre-year.

Window Settings

Optimize

Inspect

Data Preview

Output stream name *

RankMoviesByRatings

Documentation

Incoming stream *

DerivePrimaryGenre

1. Over

2. Sort

3. Range by

4. Window columns

DerivePrimaryGenre's column

Name as

abc PrimaryGenre

PrimaryGenre

abc year

year

Window Settings

Optimize

Inspect

Data Preview

Output stream name *

RankMoviesByRatings

Documentation

Incoming stream *

DerivePrimaryGenre

1. Over

2. Sort

3. Range by

4. Window columns

DerivePrimaryGenre's column

Order

Nulls first

abc Rotten Tomato

↓

☒

+

Window Settings Optimize Inspect Data Preview ●

Output stream name * [Documentation](#)

Incoming stream * [DerivePrimaryGenre](#)

1. Over 2. Sort **3. Range by** 4. Window columns

Option * ☒ Range by current row offset ☐ Range by column value

Unbounded ☒

Window Settings Optimize Inspect Data Preview ●

Output stream name * [Documentation](#)

Incoming stream * [DerivePrimaryGenre](#)

1. Over 2. Sort 3. Range by **4. Window columns**

123

5. **Aggregate ratings with an Aggregate Transformation.** Now that you have gathered and derived all your required data, we can add an [Aggregate transformation](#) to calculate metrics based on a desired group by clicking on the **+ icon** next to your Window transformation and clicking Aggregate under Schema modifier. As you did in the window transformation, let's group movies by PrimaryGenre and year

Aggregate settings [Optimize](#) [Inspect](#) [Data preview](#) ●

Output stream name * [Learn more](#)

Incoming stream *

[Group by](#) [Aggregates](#)

RankMoviesByRatings's column	Name as	
<input type="text" value="PrimaryGenre"/>	<input type="text" value="PrimaryGenre"/>	+
<input type="text" value="year"/>	<input type="text" value="year"/>	+

In the Aggregates tab, you can aggregations calculated over the specified group by columns. For every genre and year, lets get the average Rotten Tomatoes rating, the highest and lowest rated movie (utilizing the windowing function) and the number of movies that are in each group. Aggregation significantly reduces the number of rows in your transformation stream and only propagates the group by and aggregate columns specified in the transformation.

Aggregate settings [Optimize](#) [Inspect](#) [Data preview](#) ● [Description](#) ^

Output stream name * [Learn more](#)

Incoming stream *

[Group by](#) [Aggregates](#)

Grouped by: PrimaryGenre, year

AverageRating	<input type="text" value="avg(toInteger(Rotten Tomato))"/>	1.2	+
HighestRated	<input type="text" value="first(title)"/>	abc	+
LowestRated	<input type="text" value="last(title)"/>	abc	+
NumberOfMovies	<input type="text" value="count()"/>	121	+

- To see how the aggregate transformation changes your data, use the Data Preview tab

6. **Specify Upsert condition via an Alter Row Transformation.** If you are writing to a tabular sink, you can specify insert, delete, update and upsert policies on rows using the [Alter Row transformation](#) by clicking on the + icon next to your Aggregate transformation and clicking Alter Row under Row modifier. Since you are always inserting and updating, you can specify that all rows will always be upserted.

Alter row settings Optimize Inspect Data preview ●

Output stream name * UpsertIfTrue [Learn more](#)

Incoming stream * AggregateRatings

Alter row conditions * Upsert if true()

Task 4: Writing to a Data Sink

1. **Write to a Azure Synapse Analytics Sink.** Now that you have finished all your transformation logic, you are ready to write to a Sink.
 1. Add a **Sink** by clicking on the **+** icon next to your Upsert transformation and clicking Sink under Destination.
 2. In the Sink tab, create a new data warehouse dataset via the **+ New button**.
 3. Select **Azure Synapse Analytics** from the tile list.
 4. Select a new linked service and configure your Azure Synapse Analytics connection to connect to the DWDB database. Click **Create** when finished.

New linked service (Azure Synapse Analytics (formerly SQL DW))

Name * AzureSynapseAnalytics1

Description

Connect via integration runtime * AutoResolveIntegrationRuntime

Connection string Azure Key Vault

Account selection method ☒ From Azure subscription ☐ Enter manually

Azure subscription

Server name * dwhservicecto

Database name * DWDB

Authentication type * SQL authentication

User name * ctoadmin

Password Azure Key Vault

Additional connection properties

+ New

Annotations

+ New

> Parameters

> Advanced ●

5. In the dataset configuration, select **Create new table** and enter in the schema of **Dbo** and the table name of **Ratings**. Click **OK** once completed.

Set properties

Name

AzureSynapseAnalyticsTable1

Linked service *

AzureSynapseAnalytics1

[Edit connection](#)

☐ Select from existing table ☒ Create new table

Schema and table name

dbo

Ratings

▶ Advanced

6. Since an upsert condition was specified, you need to go to the Settings tab and select 'Allow upsert' based on key columns PrimaryGenre and year.

[Sink](#) [Settings](#) [Mapping](#) [Optimize](#) [Inspect](#) [Data preview](#) ●

Update method

☐ Allow insert
☐ Allow delete
☒ Allow upsert
☐ Allow update

Key columns * ⓘ

PrimaryGenre

+

🗑

year

+

🗑

Skip writing key columns

☐

Table action

☒ None ☐ Recreate table ☐ Truncate table

Enable staging

☒

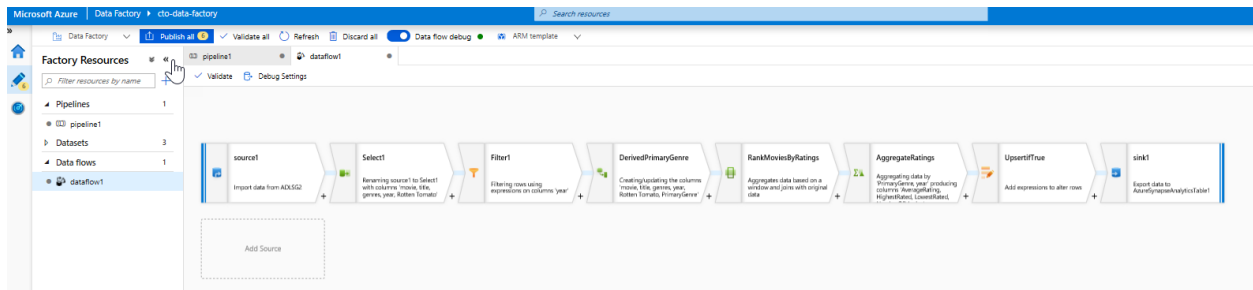
Batch size

ⓘ

Pre SQL scripts

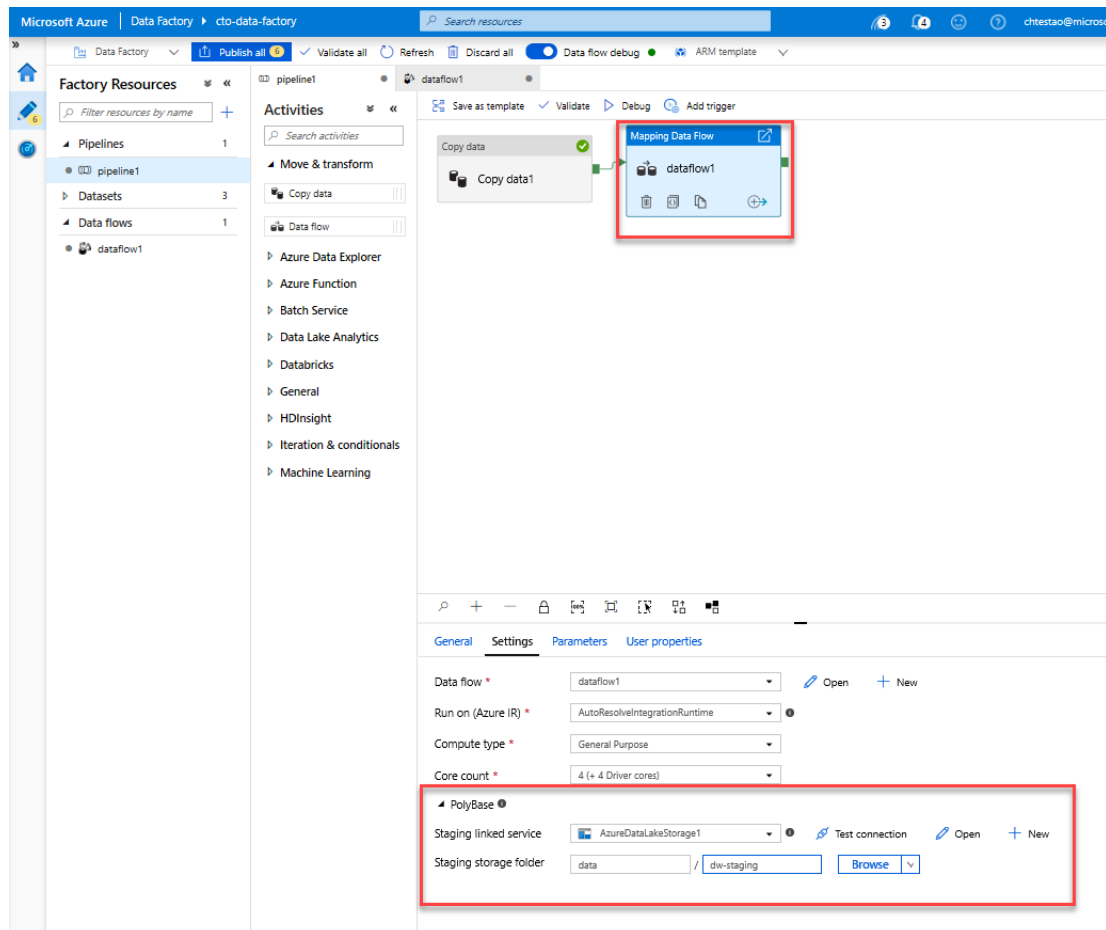
Post SQL scripts

At this point, You have finished building your 8 transformation Mapping Data Flow. It's time to run the pipeline and see the results!



Task 5: Running the Pipeline

1. Go to the pipeline1 tab in the canvas. Because Azure Synapse Analytics in Data Flow uses [PolyBase](#), you must specify a blob or ADLS staging folder. In the Execute Data Flow activity's settings tab, open up the PolyBase accordion and select your ADLS linked service and specify a staging folder path.



2. Before you publish your pipeline, run another debug run to confirm it's working as expected. Looking at the Output tab, you can monitor the status of both activities as they are running.
3. Once both activities succeeded, you can click on the eyeglasses icon next to the Data Flow activity to get a more in depth look at the Data Flow run.
4. If you used the same logic described in this lab, your Data Flow will write 737 rows to your SQL DW. You can go into [SQL Server Management Studio](#) to verify the pipeline worked correctly and see what got written.

SQLQuery1.sql - dw...tosqladmin (2184))* Object Explorer Details

```
Select count(*) as TotalCount from dbo.Ratings
```

Select * from dbo.Ratings

100 %

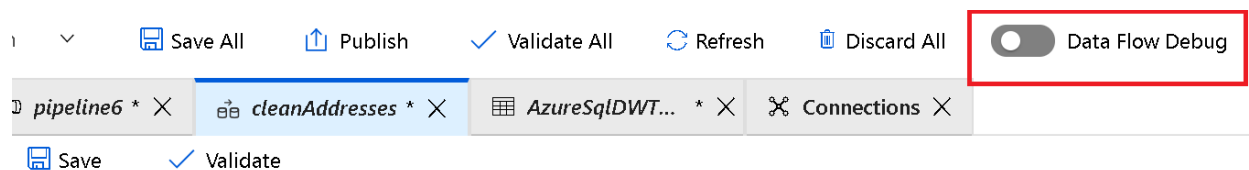
Results Messages

	TotalCount
1	737

	PrimaryGenre	year	AverageRating	HighestRated	LowestRated	NumberOfMovies
1	Action	1955	82.5	Dam Busters, The	To Hell and Back	4
2	(no genres listed)	1994	83	Freaky Friday	Freaky Friday	2
3	(no genres listed)	2012	63	Doctor Who: The Time of the Doctor	Doctor Who: The Time of the Doctor	2
4	Thriller	1963	51	Seven Days in May	Seven Days in May	2
5	(no genres listed)	2009	50	Boy Crazy	Boy Crazy	2
6	(no genres listed)	1964	89	Scorpio Rising	Scorpio Rising	2

Mapping data flow Debug Mode

During the building of Mapping Data Flows, you can interactively watch how the data transformations are executing so that you can debug them. To use this functionality, it is first necessary to turn on the "Data Flow Debug" feature.



Clicking Debug will provision the Spark clusters required to interact with the Mapping Data Flow transformations. On turning Debug on, you will be prompted to select the Integration Runtime that you require to use in the environment. If you select AutoResolveIntegrationRuntime, a cluster with eight cores that will be available with a time to live value of 60 minutes.

A Data Preview tab is available in Debug mode that will allow you to view the data at each stage of the pipeline. You can view the data after each transformation. The data previewer also provides the ability to actions on the data such as looking at descriptive statistics of the data, or the ability to modify the data.

Derived Column's Settings	Optimize	Inspect	Data Preview	Describe
Number of rows	INSERT 100	UPDATE 0	DELETE 0	UPSERT 0
LOOKUP 0	TOTAL 1000			
abc	Typecast	Modify	Statistics	Remove
movieid	abc	title	abc	genres
year	abc			
1		Toy Story (1995)	Adventure Animation Children Comedy Fantasy	1995
2		Jumanji (1995)	Adventure Children Fantasy	1995
3		Grumpier Old Men (1995)	Comedy Romance	1995
4		Waiting to Exhale (1995)	Comedy Drama Romance	1995
5		Father of the Bride Part II (1995)	Comedy	1995
6		Heat (1995)	Action Crime Thriller	1995
7		Sabrina (1995)	Comedy Romance	1995
8		Tom and Huck (1995)	Adventure Children	1995
9		Sudden Death (1995)	Action	1995

Finally, you can use the debug settings to control the number of rows that are returned within the data previewer.

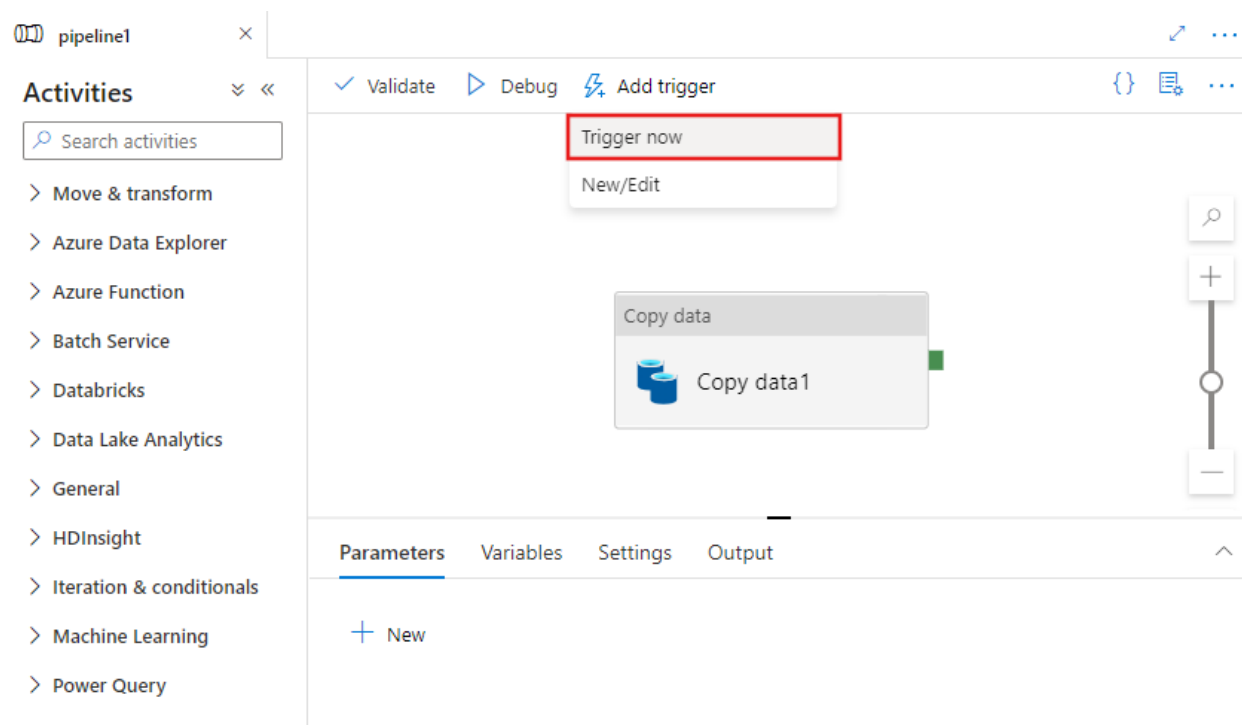
Pipeline execution and Triggers in Azure Data Factory

A *pipeline run* in Azure Data Factory and Azure Synapse defines an instance of a pipeline execution. For example, say you have a pipeline that executes at 8:00 AM, 9:00 AM, and 10:00 AM. In this case, there are three separate runs of the pipeline or pipeline runs. Each pipeline run has a unique pipeline run ID. A run ID is a GUID that uniquely defines that particular pipeline run.

Pipeline runs are typically instantiated by passing arguments to parameters that you define in the pipeline. You can execute a pipeline either manually or by using a *trigger*. This article provides details about both ways of executing a pipeline.

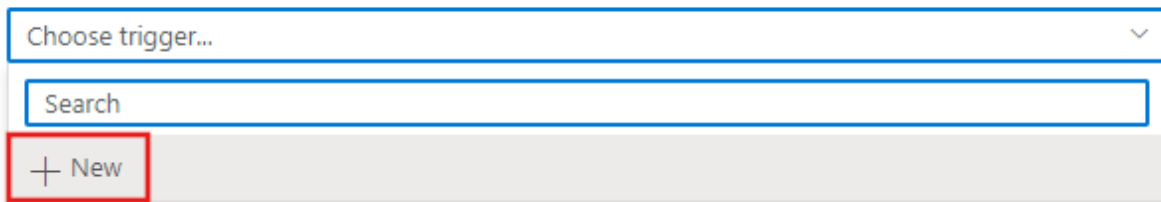
Create triggers with UI

To manually trigger a pipeline or configure a new scheduled, tumbling window, storage event, or custom event trigger, select Add trigger at the top of the pipeline editor.



If you choose to manually trigger the pipeline, it will execute immediately. Otherwise if you choose New/Edit, you will be prompted with the add triggers window to either choose an existing trigger to edit, or create a new trigger.

Add triggers



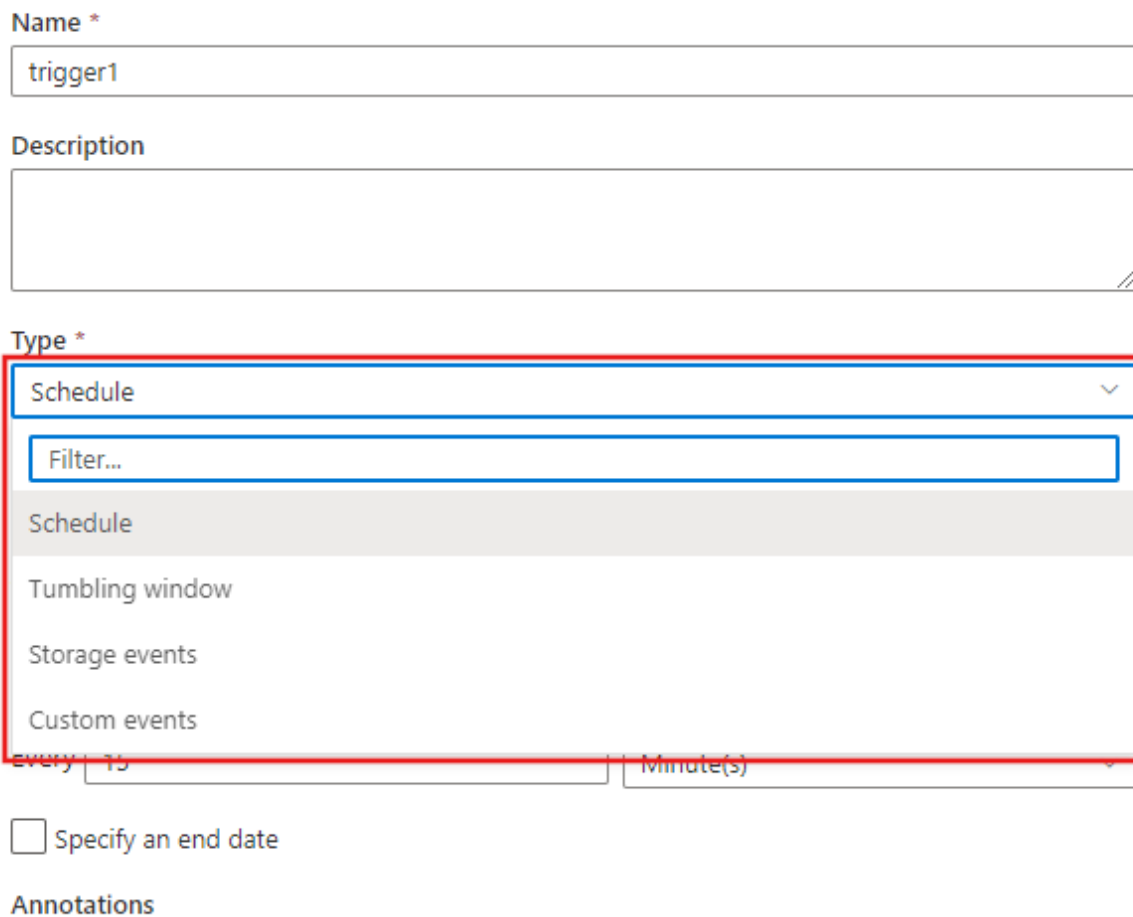
Choose trigger... ▾

Search

+ New

You will see the trigger configuration window, allowing you to choose the trigger type.

New trigger



Name *

trigger1

Description

Type *

Schedule ▾

Filter...

Schedule

Tumbling window

Storage events

Custom events

every 15 Minute(s)

☐ Specify an end date

Annotations

OK

Cancel

Triggers are another way that you can execute a pipeline run. Triggers represent a unit of processing that determines when a pipeline execution needs to be kicked off. Currently, the service supports three types of triggers:

- Schedule trigger: A trigger that invokes a pipeline on a wall-clock schedule.
- Tumbling window trigger: A trigger that operates on a periodic interval, while also retaining state.
- Event-based trigger: A trigger that responds to an event.

Pipelines and triggers have a many-to-many relationship (except for the tumbling window trigger). Multiple triggers can kick off a single pipeline, or a single trigger can kick off multiple pipelines.

Schedule trigger with JSON

A schedule trigger runs pipelines on a wall-clock schedule. This trigger supports periodic and advanced calendar options. For example, the trigger supports intervals like "weekly" or "Monday at 5:00 PM and Thursday at 9:00 PM." The schedule trigger is flexible because the dataset pattern is agnostic, and the trigger doesn't discern between time-series and non-time-series data.

Tumbling window trigger

Tumbling window triggers are a type of trigger that fires at a periodic time interval from a specified start time, while retaining state. Tumbling windows are a series of fixed-sized, non-overlapping, and contiguous time intervals.

Event-based trigger

An event-based trigger runs pipelines in response to an event. There are two flavors of event-based triggers.

- *Storage event trigger* runs a pipeline against events happening in a Storage account, such as the arrival of a file, or the deletion of a file in Azure Blob Storage account.
- *Custom event trigger* processes and handles [custom articles](#) in Event Grid

Trigger type comparison

The tumbling window trigger and the schedule trigger both operate on time heartbeats. How are they different?

The following table provides a comparison of the tumbling window trigger and schedule trigger:

TRIGGER TYPE COMPARISON		
Item	Tumbling window trigger	Schedule trigger
Backfill scenarios	Supported. Pipeline runs can be scheduled for windows in the past.	Not supported. Pipeline runs can be executed only on time periods from the current time and the future.
Reliability	100% reliability. Pipeline runs can be scheduled for all windows from a specified start date without gaps.	Less reliable.
Retry capability	Supported. Failed pipeline runs have a default retry policy of 0, or a policy that's specified by the user in the trigger definition. Automatically retries when the pipeline runs fail due to concurrency/server/throttling limits (that is, status codes 400: User Error, 429: Too many requests, and 500: Internal Server error).	Not supported.
Concurrency	Supported. Users can explicitly set concurrency limits for the trigger. Allows between 1 and 50 concurrent triggered pipeline runs.	Not supported.
System variables	Along with <code>@trigger().scheduledTime</code> and <code>@trigger().startTime</code> , it also supports the use of the WindowStart and WindowEnd system variables. Users can access <code>trigger().outputs.windowStartTime</code> and <code>trigger().outputs.windowEndTime</code> as trigger system variables in the trigger definition. The values are used as the window start time and window end time, respectively. For example, for a tumbling window trigger that runs every hour, for the window 1:00 AM to 2:00 AM, the definition is <code>trigger().outputs.windowStartTime = 2017-09-01T01:00:00Z</code> and <code>trigger().outputs.windowEndTime = 2017-09-01T02:00:00Z</code> .	Only supports default <code>@trigger().scheduledTime</code> and <code>@trigger().startTime</code> variables.

Pipeline-to-trigger relationship	Supports a one-to-one relationship. Only one pipeline can be triggered.	Supports many-to-many relationships. Multiple triggers can kick off a single pipeline. A single trigger can kick off multiple pipelines.
---	---	--

Control Flow:

1. **Append Variable activity:** Use the Append Variable activity to add a value to an existing array variable defined in a Data Factory or Synapse Analytics pipeline.
2. **Set Variable:** Use the Set Variable activity to set the value of an existing variable of type String, Bool, or Array defined in a Data Factory or Synapse pipeline.
3. **Execute Pipeline:** The Execute Pipeline activity allows a Data Factory or Synapse pipeline to invoke another pipeline.
4. **Fail Activity:** You might occasionally want to throw an error in a pipeline intentionally. A Lookup activity might return no matching data, or a Custom activity might finish with an internal error. Whatever the reason might be, now you can use a Fail activity in a pipeline and customize both its error message and error code.
5. **Filter:** You can use a Filter activity in a pipeline to apply a filter expression to an input array.
6. **ForEach:** The ForEach Activity defines a repeating control flow in an Azure Data Factory or Synapse pipeline. This activity is used to iterate over a collection and executes specified activities in a loop. The loop implementation of this activity is similar to Foreach looping structure in programming languages.
7. **Metadata:** You can use the Get Metadata activity to retrieve the metadata of any data in Azure Data Factory or a Synapse pipeline. You can use the output from the Get Metadata activity in conditional expressions to perform validation, or consume the metadata in subsequent activities.
8. **If Condition:** The If Condition activity provides the same functionality that an if statement provides in programming languages. It executes a set of activities when the condition evaluates to true and another set of activities when the condition evaluates to false.
9. **Lookup activity** can retrieve a dataset from any of the data sources supported by data factory and Synapse pipelines. You can use it to dynamically determine which objects to operate on in a subsequent activity, instead of hard coding the object name. Some object examples are files and tables.

Lookup activity reads and returns the content of a configuration file or table. It also returns the result of executing a query or stored procedure. The output can be a

singleton value or an array of attributes, which can be consumed in a subsequent copy, transformation, or control flow activities like ForEach activity.

10. Use the Set Variable activity to set the value of an existing variable of type String, Bool, or Array defined in a Data Factory or Synapse pipeline.
11. The Switch activity provides the same functionality that a switch statement provides in programming languages. It evaluates a set of activities corresponding to a case that matches the condition evaluation.
12. The Until activity provides the same functionality that a do-until looping structure provides in programming languages. It executes a set of activities in a loop until the condition associated with the activity evaluates to true. You can specify a timeout value for the until activity.
13. You can use a Validation in a pipeline to ensure the pipeline only continues execution once it has validated the attached dataset reference exists, that it meets the specified criteria, or timeout has been reached.
14. When you use a Wait activity in a pipeline, the pipeline waits for the specified period of time before continuing with execution of subsequent activities.
15. Web Activity can be used to call a custom REST endpoint from an Azure Data Factory or Synapse pipeline. You can pass datasets and linked services to be consumed and accessed by the activity.
16. A webhook activity can control the execution of pipelines through custom code. With the webhook activity, code can call an endpoint and pass it a callback URL. The pipeline run waits for the callback invocation before it proceeds to the next activity.

Data Flow:

Data flows are available both in Azure Data Factory and Azure Synapse Pipelines. This article applies to mapping data flows. If you are new to transformations, please refer to the introductory article [Transform data using a mapping data flow](#).

Below is a list of the transformations currently supported in mapping data flow. Click on each transformations to learn its configuration details.

Name	Category	Description
Aggregate	Schema modifier	Define different types of aggregations such as SUM, MIN, MAX, and COUNT grouped by existing or computed columns.
Alter row	Row modifier	Set insert, delete, update, and upsert policies on rows.
Assert	Row modifier	Set assert rules for each row.

<u>Conditional split</u>	Multiple inputs/outputs	Route rows of data to different streams based on matching conditions.
<u>Derived column</u>	Schema modifier	Generate new columns or modify existing fields using the data flow expression language.
<u>External call</u>	Schema modifier	Call external endpoints inline row-by-row.
<u>Exists</u>	Multiple inputs/outputs	Check whether your data exists in another source or stream.
<u>Filter</u>	Row modifier	Filter a row based upon a condition.
<u>Flatten</u>	Formatters	Take array values inside hierarchical structures such as JSON and unroll them into individual rows.
<u>Join</u>	Multiple inputs/outputs	Combine data from two sources or streams.
<u>Lookup</u>	Multiple inputs/outputs	Reference data from another source.
<u>New branch</u>	Multiple inputs/outputs	Apply multiple sets of operations and transformations against the same data stream.
<u>Parse</u>	Formatters	Parse text columns in your data stream that are strings of JSON, delimited text, or XML formatted text.
<u>Pivot</u>	Schema modifier	An aggregation where one or more grouping columns has its distinct row values transformed into individual columns.
<u>Rank</u>	Schema modifier	Generate an ordered ranking based upon sort conditions
<u>Select</u>	Schema modifier	Alias columns and stream names, and drop or reorder columns
<u>Sink</u>	-	A final destination for your data
<u>Sort</u>	Row modifier	Sort incoming rows on the current data stream
<u>Source</u>	-	A data source for the data flow
<u>Stringify</u>	Formatters	Turn complex types into plain strings
<u>Surrogate key</u>	Schema modifier	Add an incrementing non-business arbitrary key value
<u>Union</u>	Multiple inputs/outputs	Combine multiple data streams vertically
<u>Unpivot</u>	Schema modifier	Pivot columns into row values
<u>Window</u>	Schema modifier	Define window-based aggregations of columns in your data streams.

Parameterize:

1. Parameterizing Linked Service:

You can now parameterize a linked service and pass dynamic values at run time. For example, if you want to connect to different databases on the same logical SQL server, you can now parameterize the database name in the linked service definition. This prevents you from having to create a linked service for each database on the logical SQL server. You can parameterize other properties in the linked service definition as well - for example, User name.

2. Global Parameters:

Global parameters are constants across a data factory that can be consumed by a pipeline in any expression. They're useful when you have multiple pipelines with identical parameter names and values. When promoting a data factory using the continuous integration and deployment process (CI/CD), you can override these parameters in each environment.

3. System Variables:

Pipeline scope

These system variables can be referenced anywhere in the pipeline JSON.

PIPELINE SCOPE	
Variable Name	Description
@pipeline().DataFactory	Name of the data or Synapse workspace the pipeline run is running in
@pipeline().Pipeline	Name of the pipeline
@pipeline().RunId	ID of the specific pipeline run
@pipeline().TriggerType	The type of trigger that invoked the pipeline (for example, <code>ScheduleTrigger</code> , <code>BlobEventsTrigger</code>). For a list of supported trigger types, see Pipeline execution and triggers . A trigger type of <code>Manual</code> indicates that the pipeline was triggered manually.
@pipeline().TriggerId	ID of the trigger that invoked the pipeline
@pipeline().TriggerName	Name of the trigger that invoked the pipeline
@pipeline().TriggerTime	Time of the trigger run that invoked the pipeline. This is the time at which the trigger actually fired to invoke the pipeline run, and it may differ slightly from the trigger's scheduled time.
@pipeline().GroupId	ID of the group to which pipeline run belongs.
@pipeline()?TriggeredByPipelineName	Name of the pipeline that triggers the pipeline run. Applicable when the pipeline run is triggered by an <code>ExecutePipeline</code> activity. Evaluate to <i>Null</i> when used in other circumstances. Note the question mark after <code>@pipeline()</code>
@pipeline()?TriggeredByPipelineRunId	Run ID of the pipeline that triggers the pipeline run. Applicable when the pipeline run is triggered by an <code>ExecutePipeline</code> activity. Evaluate to <i>Null</i> when used in other circumstances. Note the question mark after <code>@pipeline()</code>

Schedule trigger scope

These system variables can be referenced anywhere in the trigger JSON for triggers of type [ScheduleTrigger](#).

SCHEDULE TRIGGER SCOPE	
Variable Name	Description
@trigger().scheduledTime	Time at which the trigger was scheduled to invoke the pipeline run.
@trigger().startTime	Time at which the trigger actually fired to invoke the pipeline run. This may differ slightly from the trigger's scheduled time.

Tumbling window trigger scope

These system variables can be referenced anywhere in the trigger JSON for triggers of type [TumblingWindowTrigger](#).

TUMBLING WINDOW TRIGGER SCOPE	
Variable Name	Description
@trigger().outputs.windowStartTime	Start of the window associated with the trigger run.
@trigger().outputs.windowEndTime	End of the window associated with the trigger run.
@trigger().scheduledTime	Time at which the trigger was scheduled to invoke the pipeline run.
@trigger().startTime	Time at which the trigger actually fired to invoke the pipeline run. This may differ slightly from the trigger's scheduled time.

Storage event trigger scope

These system variables can be referenced anywhere in the trigger JSON for triggers of type [BlobEventsTrigger](#).

STORAGE EVENT TRIGGER SCOPE	
Variable Name	Description
@triggerBody().fileName	Name of the file whose creation or deletion caused the trigger to fire.
@triggerBody().folderPath	Path to the folder that contains the file specified by @triggerBody().fileName. The first segment of the folder path is the name of the Azure Blob Storage container.
@trigger().startTime	Time at which the trigger fired to invoke the pipeline run.

Custom event trigger scope

These system variables can be referenced anywhere in the trigger JSON for triggers of type CustomEventsTrigger.

Variable Name	Description
@triggerBody().event.eventType	Type of events that triggered the Custom Event Trigger run. Event type is customer-defined field and take on any values of string type.
@triggerBody().event.subject	Subject of the custom event that caused the trigger to fire.
@triggerBody().event.data._keyName_	Data field in custom event is a free from JSON blob, which customer can use to send messages and data. Please use <i>data.keyName</i> to reference each field. For example, @triggerBody().event.data.callback returns the value for the <i>callback</i> field stored under <i>data</i> .
@trigger().startTime	Time at which the trigger fired to invoke the pipeline run.

4. Parameterizing Mapping data Flows:

Mapping data flows in Azure Data Factory and Synapse pipelines support the use of parameters. Define parameters inside of your data flow definition and use them throughout your expressions. The parameter values are set by the calling pipeline via the Execute Data Flow activity. You have three options for setting the values in the data flow activity expressions:

- Use the pipeline control flow expression language to set a dynamic value
- Use the data flow expression language to set a dynamic value
- Use either expression language to set a static literal value

Use this capability to make your data flows general-purpose, flexible, and reusable. You can parameterize data flow settings and expressions with these parameters.

Integration Runtime in Azure Data Factory

The Integration Runtime (IR) is the compute infrastructure used by Azure Data Factory and Synapse pipelines to provide data integration capabilities across different network environments.

Azure Integration Runtime:

Azure IR provides a fully managed compute to natively perform data movement and dispatch data transformation activities to compute services like HDInsight. It is hosted in Azure environment and supports connecting to resources in public network environment with public accessible endpoints.

Self-hosted IR:

A self-hosted integration runtime can run copy activities between a cloud data store and a data store in a private network. It also can dispatch transform activities against compute resources in an on-premises network or an Azure virtual network. The installation of a self-hosted integration runtime needs an on-premises machine or a virtual machine inside a private network.

Azure SSIS IR:

Azure-SQL Server Integration Services (SSIS) integration runtime (IR) in Azure Data Factory (ADF) and Azure Synapse Pipelines. An Azure-SSIS IR supports:

- Running packages deployed into SSIS catalog (SSISDB) hosted by Azure SQL Database server/Managed Instance (Project Deployment Model)
- Running packages deployed into file system, Azure Files, or SQL Server database (MSDB) hosted by Azure SQL Managed Instance (Package Deployment Model)