# **Azure Data Factory: A Complete Guide for Beginners**

Learn the basics of Azure Data Factory and build your first data pipeline with this step-by-step guide!

Jan 14, 2025 · 13 min read

**Contents**

[What is Azure Data Factory?](https://www.datacamp.com/tutorial/azure-data-factory#what-is-azure-data-factory?-%3Cahre)

[Features of Azure Data Factory](https://www.datacamp.com/tutorial/azure-data-factory#features-of-azure-data-factory-%3Cspan)

[Core Components of Azure Data Factory](https://www.datacamp.com/tutorial/azure-data-factory#core-components-of-azure-data-factory-%3Cspan)

[Setting Up Azure Data Factory](https://www.datacamp.com/tutorial/azure-data-factory#setting-up-azure-data-factory-%3Cspan)

[1. Pre-requisites](https://www.datacamp.com/tutorial/azure-data-factory#1.-pre-requisites-%3Cspan)

[2. Creating an Azure Data Factory instance](https://www.datacamp.com/tutorial/azure-data-factory#2.-creating-an-azure-data-factory-instance-%3Cspan)

[3. Navigating the ADF interface](https://www.datacamp.com/tutorial/azure-data-factory#3.-navigating-the-adf-interface-%3Cspan)

[Building Your First Pipeline in Azure Data Factory](https://www.datacamp.com/tutorial/azure-data-factory#building-your-first-pipeline-in-azure-data-factory-%3Cspan)

[Azure Data Factory Integration and Transformation Capabilities](https://www.datacamp.com/tutorial/azure-data-factory#azure-data-factory-integration-and-transformation-capabilities-%3Cspan)

[Azure Data Factory Use Cases](https://www.datacamp.com/tutorial/azure-data-factory#azure-data-factory-use-cases-%3Cspan)

[Best Practices for Using Azure Data Factory](https://www.datacamp.com/tutorial/azure-data-factory#best-practices-for-using-azure-data-factory-%3Cspan)

[Conclusion](https://www.datacamp.com/tutorial/azure-data-factory#conclusion-azure)

Azure Data Factory (ADF) is Microsoft’s cloud-based data integration service tailored for modern organizations. It empowers users to design, manage, and automate workflows that handle data movement and transformation tasks at an enterprise scale.

ADF stands out for its user-friendly, no-code interface, which allows both technical and non-technical users to [**build data pipelines**](https://www.datacamp.com/tutorial/introduction-to-data-pipelines-for-data-professionals) easily. Its extensive integration capabilities support [**over 90 native connectors**](https://learn.microsoft.com/en-us/fabric/data-factory/connector-overview), enabling data flow across diverse sources, including on-premises systems and cloud-based services.

In this guide, I offer a comprehensive introduction to Azure Data Factory, covering its components and features and providing a hands-on tutorial to help you create your first data pipeline.

## **What is Azure Data Factory?**

[**Azure Data Factory (ADF)**](https://azure.microsoft.com/en-us/products/data-factory) is a cloud-based data integration service designed to orchestrate and automate data workflows.

It is used to collect, transform, and deliver data, ensuring that insights are readily accessible for analytics and decision-making.

With its scalable and [**serverless architecture**](https://www.datacamp.com/blog/serverless-computing), ADF can handle workflows of any size—from simple data migrations to complex data transformation pipelines.

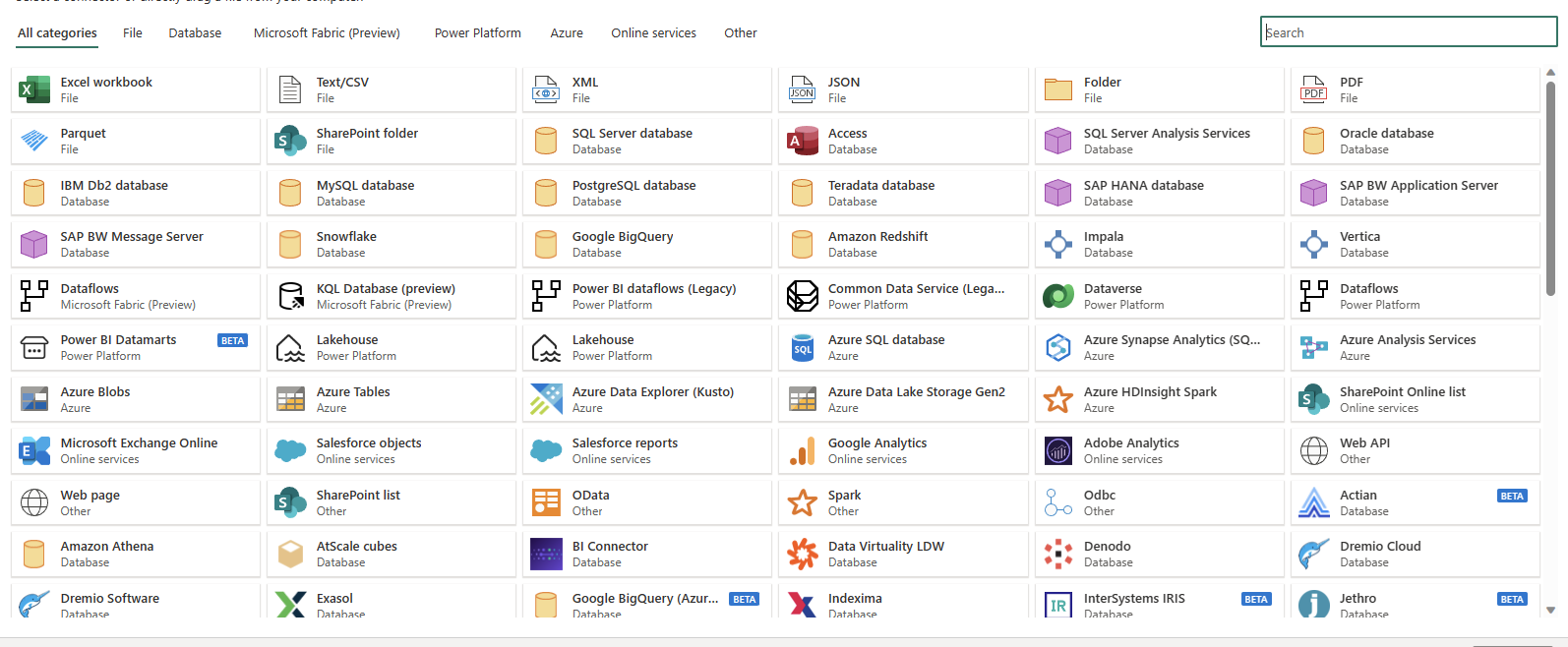
ADF bridges the gap between data silos, enabling users to move and transform data between on-premises systems, cloud services, and external platforms. Whether you’re working with big data, operational databases, or [**APIs**](https://www.datacamp.com/tutorial/python-api), Azure Data Factory provides the tools to connect, process, and unify data efficiently.

## **Features of Azure Data Factory**

Here are some of the most important features that ADF offers.

### **1. Data integration**

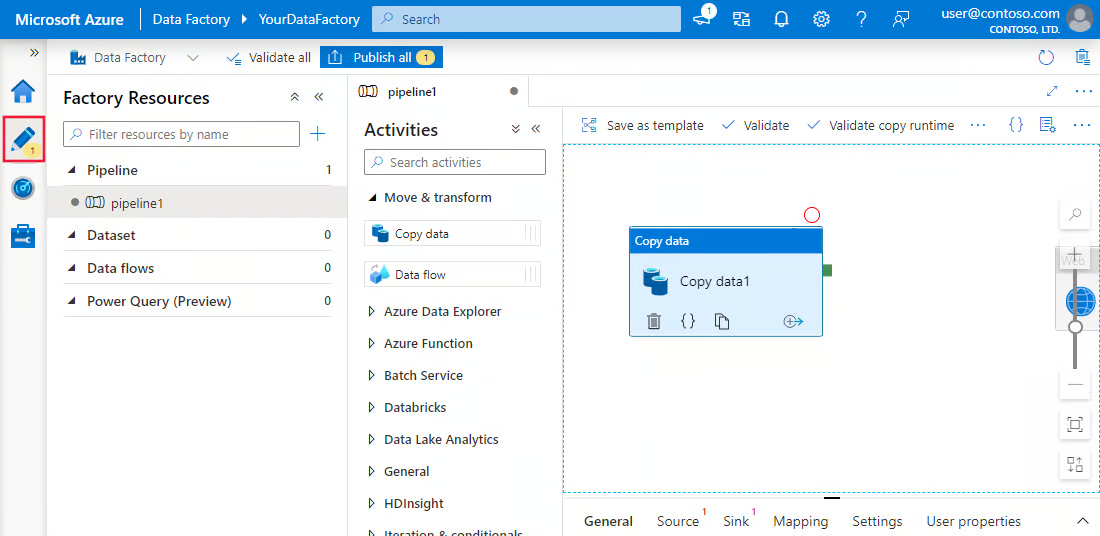
Azure Data Factory supports integration with [**over 90 data sources**](https://learn.microsoft.com/en-us/fabric/data-factory/connector-overview), including cloud-based and on-premises systems. It includes support for [**SQL databases**](https://www.datacamp.com/blog/all-about-sql-the-essential-language-for-database-management), [**NoSQL systems**](https://www.datacamp.com/tutorial/nosql-tutorial), [**REST APIs**](https://www.datacamp.com/tutorial/graphql-vs-rest), and file-based data sources, allowing you to unify data workflows regardless of the source or format.



*Data connectors available in Azure Data Factory*

### **2. No-code pipeline authoring**

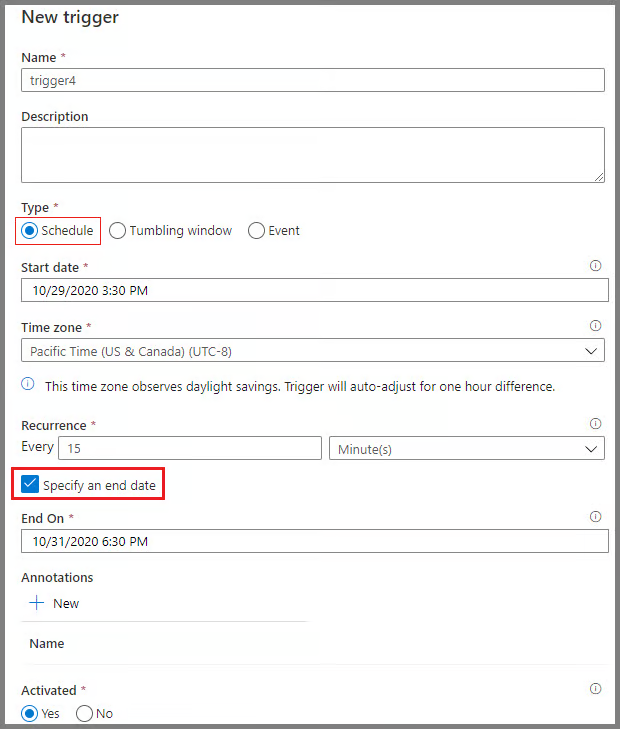
ADF’s drag-and-drop interface simplifies how users create data pipelines. With prebuilt templates, guided configuration wizards, and an intuitive visual editor, even users with no coding expertise can design comprehensive end-to-end workflows.



*No-code authoring experience in Azure Data Factory*

### **3. Scheduling**

Azure Data Factory’s scheduling tools offer workflow automation. Users can set up triggers based on specific conditions, such as a file’s arrival in cloud storage or scheduled time intervals. These scheduling options eliminate the need for manual interventions and ensure workflows are executed consistently and reliably.



*Scheduling pipelines in Azure Data Factory*

## **Core Components of Azure Data Factory**

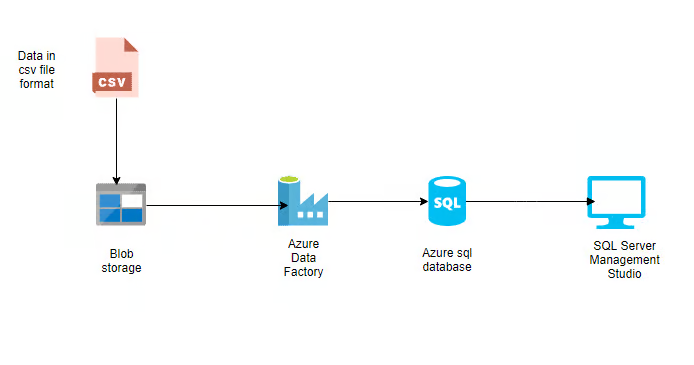
Understanding the core components of Azure Data Factory is essential to building efficient workflows.

### **1. Pipelines**

Pipelines are the backbone of Azure Data Factory. They represent data-driven workflows that define the steps required to move and transform data.

Each pipeline serves as a container for one or more activities, executed sequentially or in parallel, to achieve the desired data flow.

These pipelines enable data engineers to create end-to-end processes, such as ingesting raw data, transforming it into a usable format, and loading it into target systems.

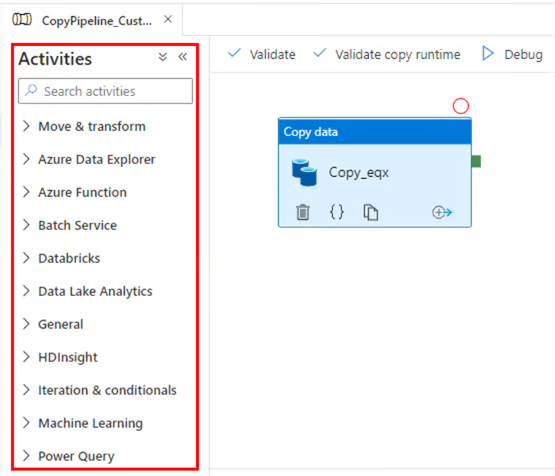


*Example of simple pipeline in Azure Data Factory*

### **2. Activities**

Activities are the functional building blocks of pipelines, each performing a specific operation. They are broadly categorized into:

* **Data movement activities:** These activities facilitate data transfer between different storage systems. For example, the "Copy data" activity moves data from Azure Blob Storage to an Azure SQL Database.
* **Data transformation activities:** These activities allow you to manipulate or process data. For instance, data flows or custom scripts can be used to transform data formats, aggregate values, or cleanse datasets.
* **Control flow activities:** These manage the logical flow of execution within pipelines. Examples include conditional branching, loops, and parallel execution, which provide flexibility in handling complex workflows.



*Activities in Azure Data Factory*

### **3. Datasets**

Datasets are representations of the data utilized in activities. They define the schema, format, and location of the data being ingested or processed.

For instance, a dataset might describe a CSV file in [**Azure Blob Storage**](https://www.datacamp.com/tutorial/guide-to-file-storage-on-azure-with-blob-and-files) or a table in an Azure SQL Database. Datasets are the intermediary layer connecting activities to the actual data sources and destinations.



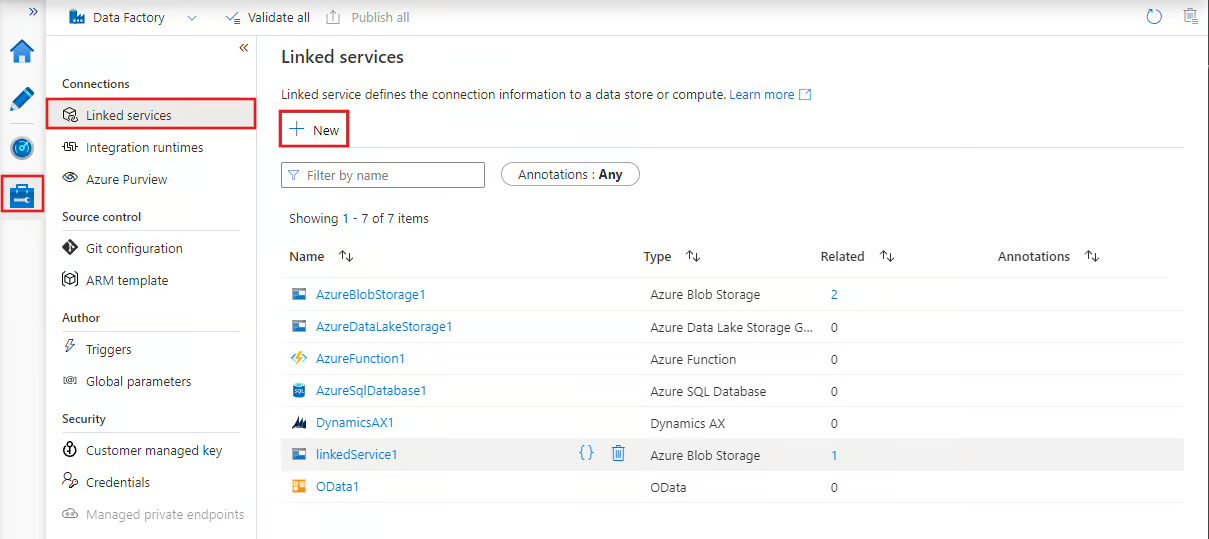
*Datasets in Azure Data Factory*

### **4. Linked services**

Linked services are connection strings that enable activities and datasets to access external systems and services.

They act as bridges between Azure Data Factory and the external resources it interacts with, such as databases, storage accounts, or compute environments.

For example, a linked service can connect to an on-premises SQL Server or a cloud-based data lake.

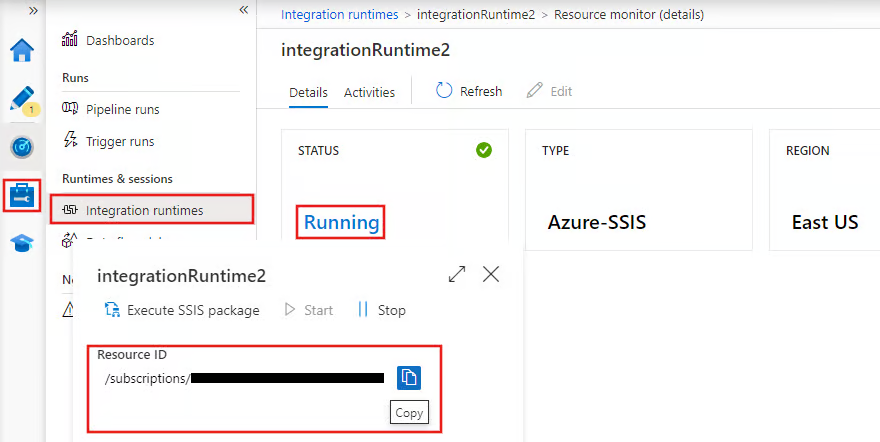


*Linked services in Azure Data Factory*

### **5. Integration runtimes**

Integration runtimes (IRs) are the compute environments that power data movement, transformation, and activity execution within Azure Data Factory. ADF provides three types of integration runtimes:

* **Azure IR:** Handles cloud-based data integration tasks and is managed entirely by Azure.
* **Self-hosted IR:** Supports data movement between on-premises systems and the cloud, making it ideal for hybrid scenarios.
* **SSIS IR:** Enables the execution of [**SQL Server Integration Services (SSIS)**](https://learn.microsoft.com/en-us/sql/integration-services/sql-server-integration-services?view=sql-server-ver16) packages within Azure, allowing you to reuse existing SSIS workflows in the cloud.



*Integration runtimes in Azure Data Factory*

## **Master Azure From Scratch**

**Build job-ready cloud computing skills by mastering Azure.**

[**Start Learning for Free**](https://www.datacamp.com/tracks/azure-fundamentals)

## **Setting Up Azure Data Factory**

Now, let’s move to the practical section of this guide!

### **1. Pre-requisites**

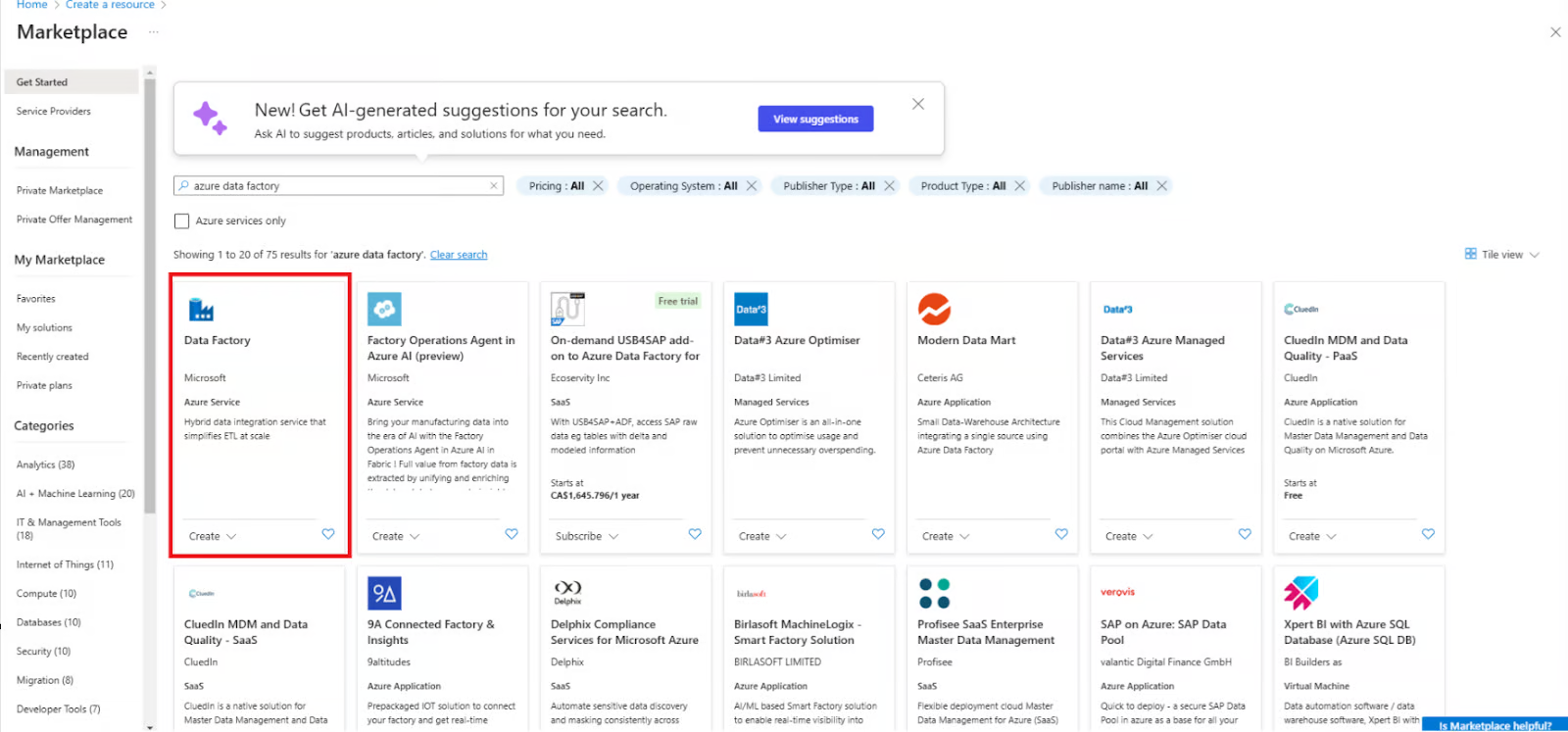
1. An active [**Azure subscription**](https://azure.microsoft.com/en-ca/pricing/purchase-options/azure-account).

2. A [**resource group**](https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/manage-resource-groups-portal) for managing Azure resources.

### **2. Creating an Azure Data Factory instance**

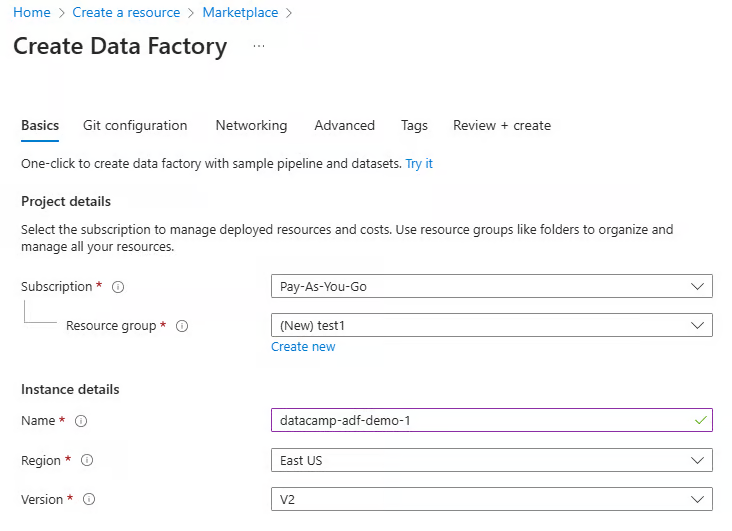
1. Log in to the [**Azure portal**](https://azure.microsoft.com/).

2. Navigate to **Create a resource** and select **Data Factory**.



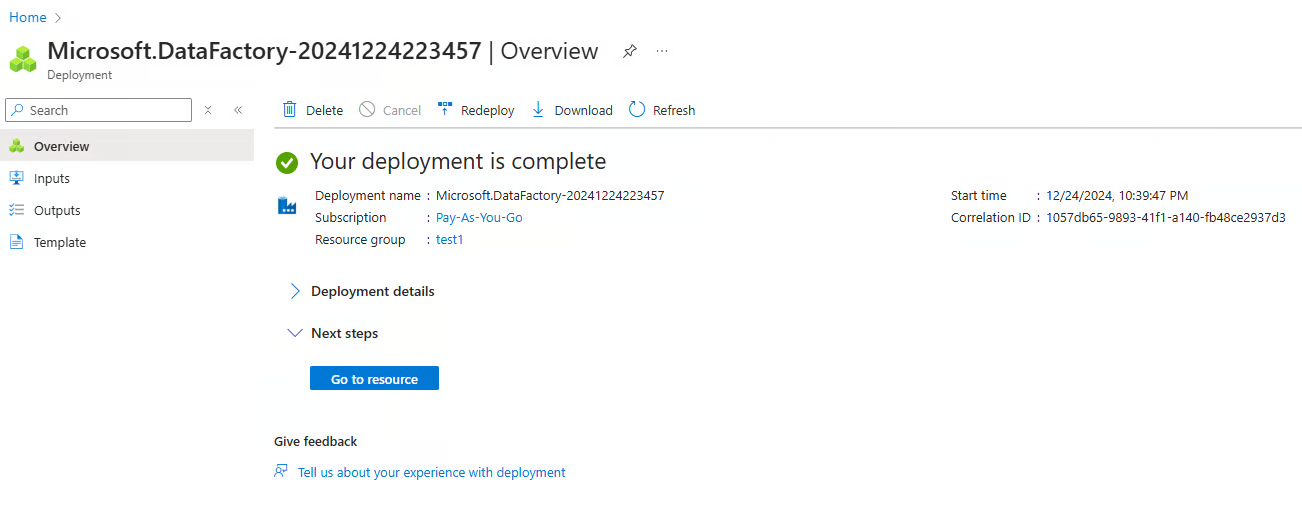
*Create a new Data Factory resource*

3. Fill out the required fields, including subscription, resource group, and region.



*Configure Data Factory resource*

4. Review and create the instance.



*Azure Data Factory instance created*

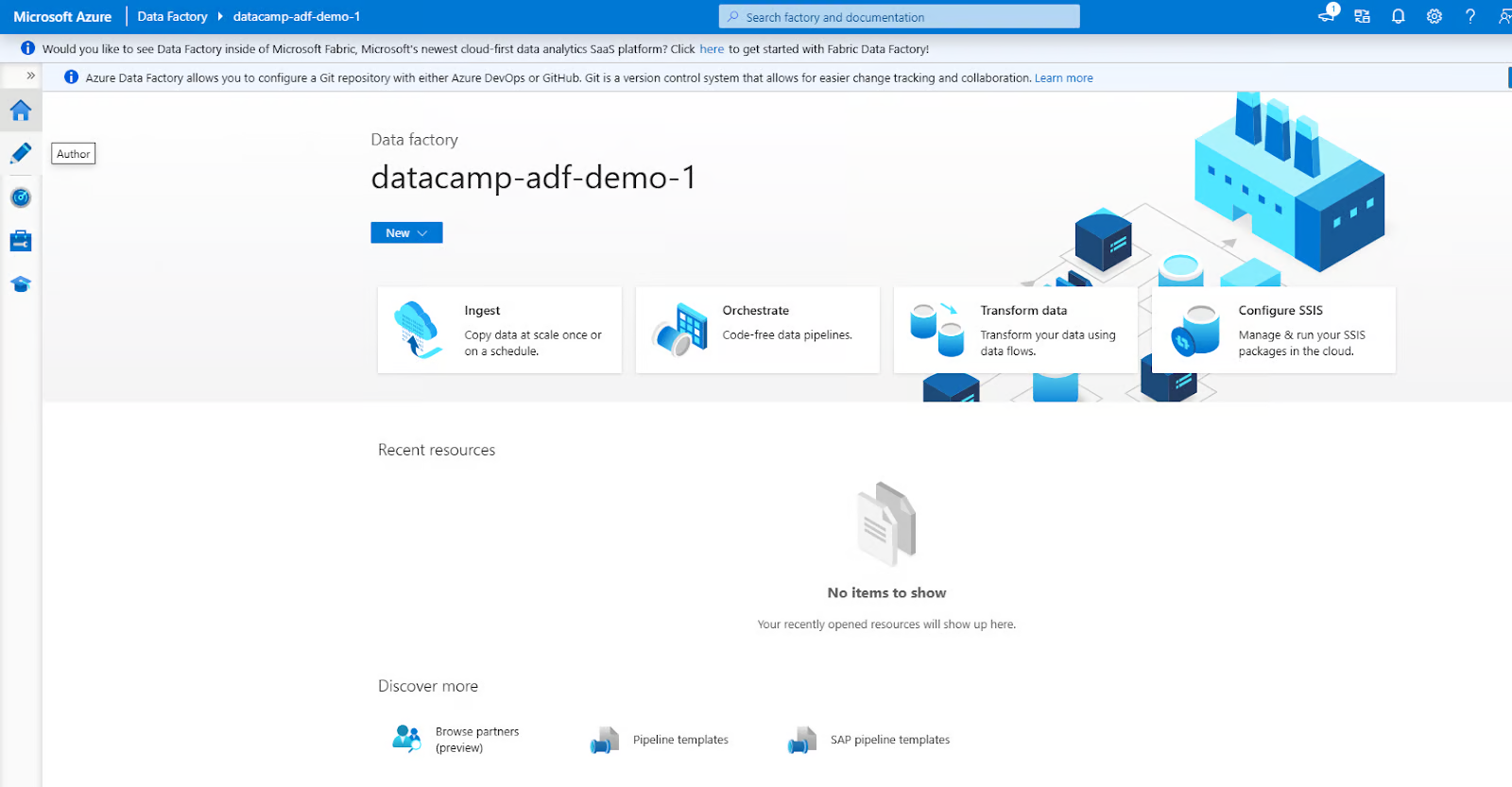
### **3. Navigating the ADF interface**

The ADF interface consists of the following main sections (accessible via the left-hand navigation menu)

**1. Author:** For creating and managing pipelines.

**2. Monitor:** To track pipeline runs and troubleshoot issues.

**3. Manage:** For configuring linked services and integration runtimes.

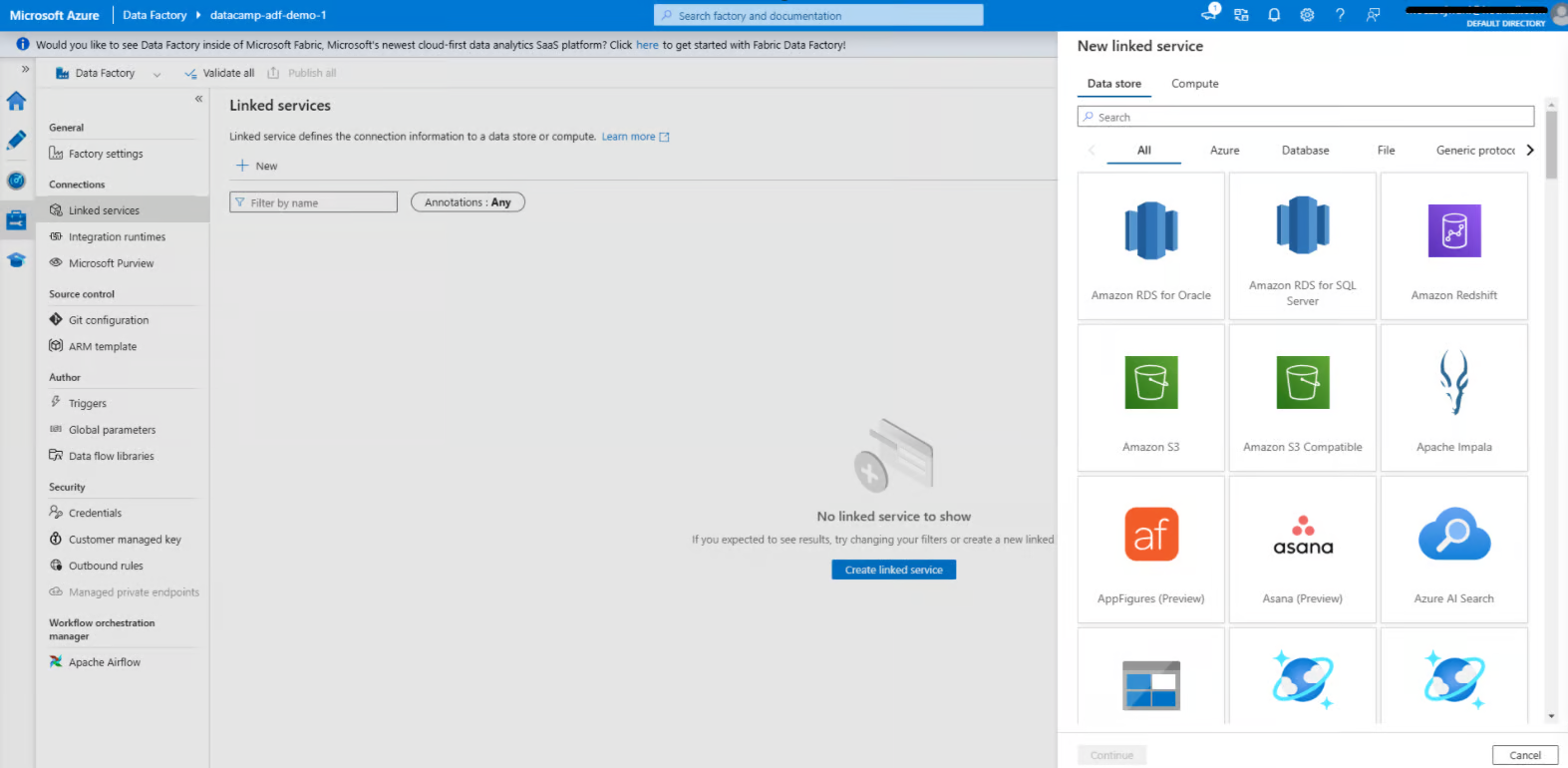


*Azure Data Factory Interface*

## **Building Your First Pipeline in Azure Data Factory**

Let’s walk through the steps to create a simple data pipeline.

### **Step 1: Create linked services**



*Creating Linked service in Azure Data Factory*

**1. Navigate to the Manage tab**

* Open your Azure Data Factory instance, and go to the **Manage** tab in the ADF interface. This is where you define linked services, which connect your data sources and destinations.

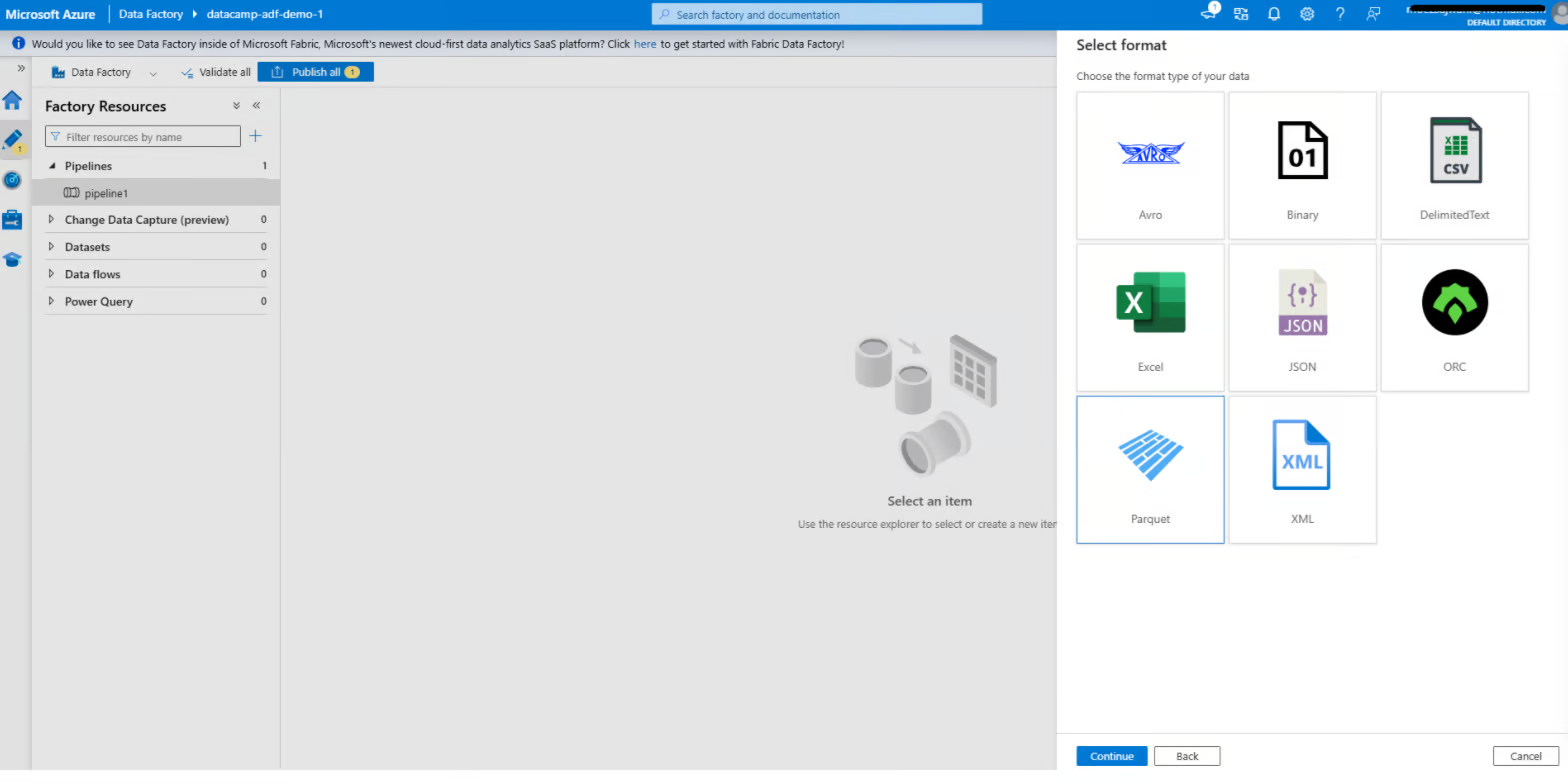
**2. Add a linked service for the data source**

* Click on **Linked services** under the Manage tab.
* Select **+ New** to create a new linked service.
* From the list of available options, select the data source you want to connect to, such as **Azure Blob** **Storage**.
* Provide the required connection details, such as the storage account name and authentication method (e.g., account key or managed identity).
* Test the connection to ensure everything is set up correctly, and click **Create**.

**3. Add a linked service for the data destination**

* Repeat the process for the data destination, such as **Azure SQL Database**.
* Select the appropriate destination type, configure the connection settings (e.g., server name, database name, and authentication method), and test the connection.
* Once verified, save the linked service.

### **Step 2: Create a dataset**



*Creating dataset in Azure Data Factory*

**1. Navigate to the Author tab**

* Open the **Author** tab in your Azure Data Factory interface. This is where you design and manage your pipelines, datasets, and other workflow components.

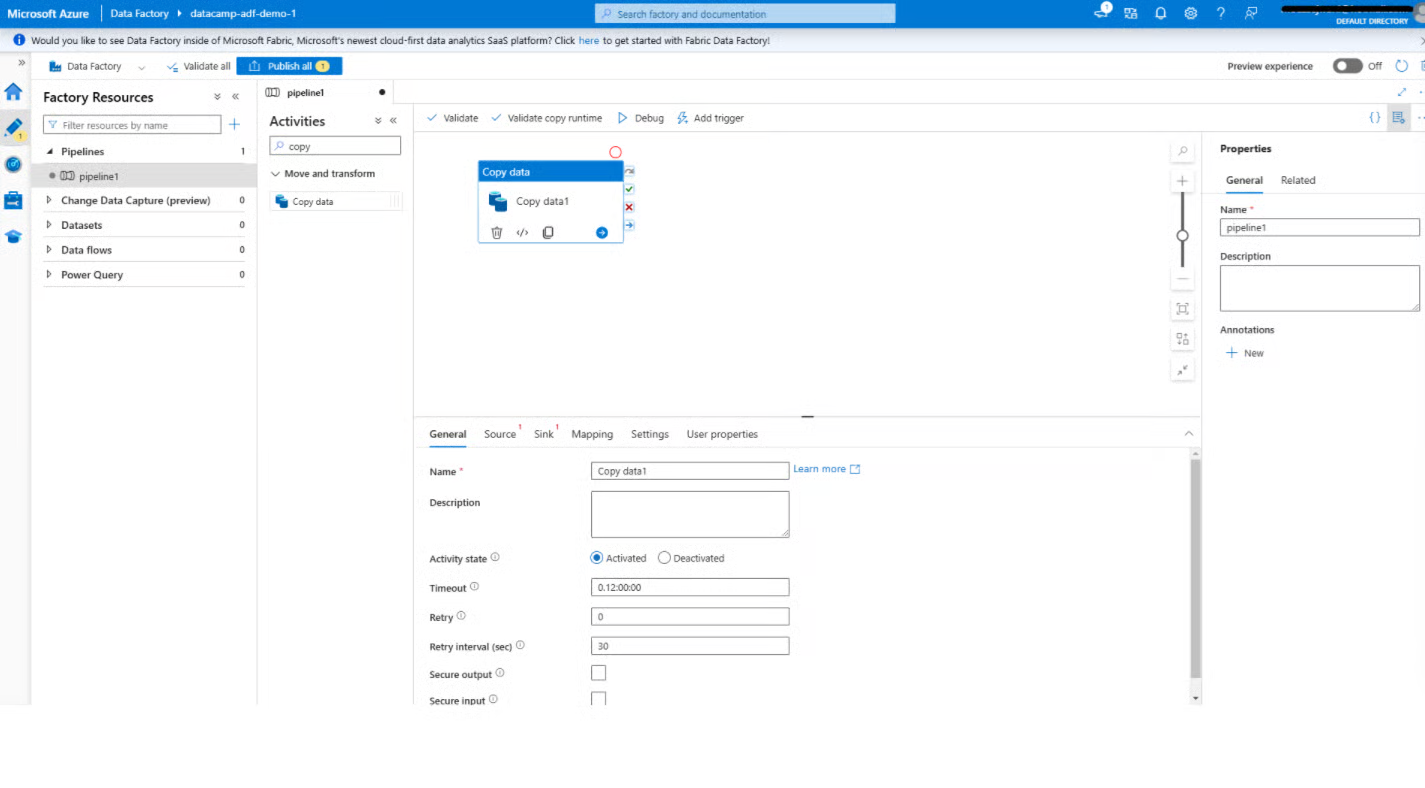
**2. Add a dataset for the source**

* Click on the **+** button and select **Dataset** from the dropdown menu.
* Choose the data store type that matches your source linked service. For example, if your source is **Azure Blob Storage**, select the corresponding data store type, such as **Delimited Text, Parquet,** or another relevant option.
* Configure the dataset:
  + **Linked service:** Select the linked service you created earlier for the data source.
  + **File path:** Specify the path or container where your source data resides.
  + **Schema and format:** Define the data format (e.g., **CSV**, **JSON**) and import the schema if applicable. This allows ADF to understand the structure of your data.
* Click **OK** to save the dataset.

**3. Add a dataset for the destination**

* Repeat the process for the destination dataset.
* Choose the data store type that matches your destination linked service. For example, if your destination is **Azure SQL Database**, select the appropriate type, such as **Table**.
* Configure the dataset:
  + **Linked service:** Select the linked service you created for the destination.
  + **Table name or path:** Specify the table or destination path where the data will be written.
  + **Schema:** Optionally define or import the schema for the destination dataset to ensure compatibility with the source data.
* Save the dataset.

### **Step 3: Add activities**



*Adding a copy data activity in Azure Data Factory*

**1. Open the Pipeline editor**

* In the **Author** tab, create a new pipeline by clicking **+** and selecting **Pipeline**.
* This will open the pipeline editor, a visual interface where you can design your data workflows.

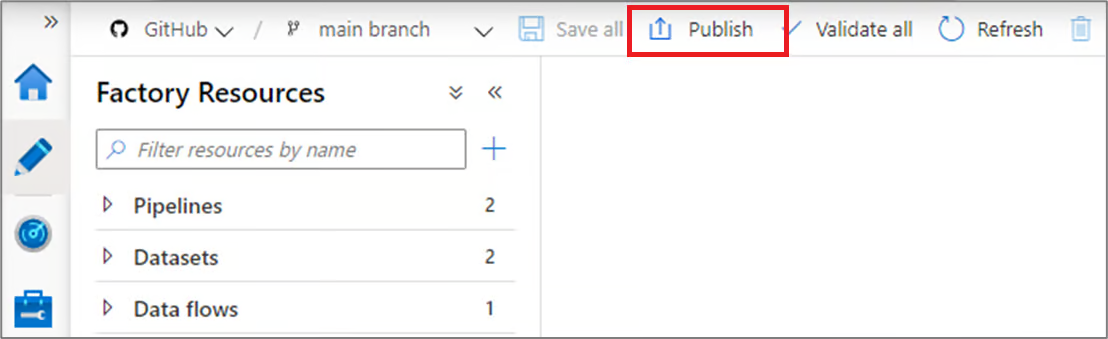
**2. Add the copy data activity**

* From the toolbox on the left, locate the **Copy data** activity under the **Move & Transform** category.
* Drag the **Copy data** activity onto the canvas. This activity moves data from the source to the destination.

**3. Configure the copy data activity**

* Click on the **Copy data** activity to open its settings pane.
* Under the **Source** tab:
  + Select the source dataset you created earlier.
  + Configure additional options such as file or folder filters if needed.
* Under the **Sink** tab:
  + Select the destination dataset.
  + Specify any additional settings, such as how to handle existing data in the destination (e.g., **overwrite** or **append**).
* Use the **Mapping** tab to align the fields or columns from the source to the destination, ensuring data compatibility.
* Save your configuration.

### **Step 4: Publish and run the pipeline**



*Publishing pipelines in Azure Data Factory*

**1. Publish your pipeline**

* Once your pipeline is configured, click **Publish** in the toolbar.
* This saves your pipeline and makes it ready for execution. Without publishing, changes made to your pipeline remain as drafts and cannot be run.

**2. Run the pipeline**

* To test your pipeline, click **Add Trigger** at the top and select **Trigger Now** for a manual run. This allows you to verify that the pipeline is functioning as expected.
* Alternatively, set up an automated schedule:
  + Go to the **Triggers** tab and create a new trigger.
  + Define the trigger conditions, such as a time-based schedule (e.g., every day at 8:00 AM) or an event-based condition (e.g., file arrival in Azure Blob Storage).
  + Associate the trigger with your pipeline to enable automation.

## **Azure Data Factory Integration and Transformation Capabilities**

Azure Data Factory offers powerful data integration and transformation features that simplify complex workflows and enhance productivity. In this section, we will review these features.

### **1. Data flows**

[**Data flows**](https://learn.microsoft.com/en-us/azure/data-factory/concepts-data-flow-overview) provide a visual environment for defining transformation logic, making it easier for users to manipulate and process data without needing to write complex code. Common tasks performed with data flows include:

* **Aggregations:** Summarize data to extract meaningful insights, such as calculating total sales or average performance metrics.
* **Joins:** Combine data from multiple sources to create enriched datasets for downstream processes.
* **Filters:** Select specific subsets of data based on defined criteria, helping to focus on relevant information.

Data flows also support advanced operations like column derivations, data type conversions, and conditional transformations, making them versatile tools for handling diverse data requirements.

### **2. Integration with Azure Synapse Analytics**

ADF integrates seamlessly with [**Azure Synapse Analytics**](https://www.datacamp.com/tutorial/azure-synapse), providing a unified platform for big data processing and advanced analytics. This integration enables users to:

* Orchestrate end-to-end data workflows that include data ingestion, preparation, and analytics.
* Leverage Synapse’s powerful query engine to process large datasets efficiently.
* Create data pipelines that feed directly into Synapse Analytics for [**machine learning**](https://www.datacamp.com/blog/what-is-machine-learning) and reporting use cases.

This synergy between ADF and Synapse helps streamline workflows and reduces the complexity of managing separate tools for data integration and analysis.

### **3. Scheduling and monitoring pipelines**

* **Scheduling:** As mentioned, ADF's scheduling capabilities offer robust automation features. Users can define triggers based on time intervals (e.g., hourly, daily) or events (e.g., the arrival of a file in [**Azure Blob Storage**](https://www.datacamp.com/tutorial/guide-to-file-storage-on-azure-with-blob-and-files)).
* **Monitoring:** The Monitor tab in Azure Data Factory, combined with [**Azure Monitor**](https://www.datacamp.com/tutorial/getting-started-with-azure-monitor), provides real-time tracking and diagnostics for pipeline executions. Users can view detailed logs, track progress, and quickly identify bottlenecks or failures. Alerts and notifications can also be configured easily.

## **Azure Data Factory Use Cases**

After an in-depth review of ADF’s features and components, let’s see what we could use it for.

### **1. Data migration**

ADF is a powerful tool for migrating data from on-premises systems to cloud-based platforms. It simplifies complex migrations by automating data movement, ensuring data integrity, and minimizing downtime.

For instance, you can use ADF to migrate data from an on-premises SQL Server to an [**Azure SQL Database**](https://www.datacamp.com/tutorial/azure-sql-database) with minimal manual intervention. By leveraging built-in connectors and integration runtimes, ADF ensures a secure and efficient migration process, accommodating both structured and unstructured data.

### **2. ETL for data warehousing**

[**Extract, transform, and load (ETL)**](https://www.datacamp.com/blog/etl-vs-elt) processes are at the heart of modern data warehousing. Azure Data Factory streamlines these workflows by integrating data from multiple sources, applying transformation logic, and loading it into a [**data warehouse**](https://www.datacamp.com/blog/data-warehouse).

For example, ADF can consolidate sales data from different regions, transform it into a unified format, and load it into Azure Synapse Analytics. This streamlined process enables you to maintain up-to-date, high-quality data for reporting and decision-making.

### **3. Data integration for data lakes**

Data lakes serve as a central repository for diverse datasets, enabling advanced analytics and machine learning. ADF facilitates ingesting data from various sources into Azure Data Lake Storage, supporting batch and streaming scenarios.

For instance, you can use ADF to collect log files, social media feeds, and IoT sensor data into a single data lake. By providing data transformation and integration tools, ADF ensures the data lake is well-organized and ready for downstream analytics and AI workloads.

## **Best Practices for Using Azure Data Factory**

Lastly, it’s worth reviewing some best practices for using ADF effectively.

### **1. Modular pipeline design**

To create maintainable and scalable workflows, design pipelines with reusable components. Modular design allows for easier debugging, testing, and updating of individual pipeline sections. For example, instead of embedding data transformation logic in every pipeline, create a reusable pipeline that can be invoked across multiple workflows. This reduces redundancy and enhances consistency across projects.

### **2. Optimize data movement**

* **Use compression:** To minimize data transfer times and reduce network bandwidth usage, compress large datasets before moving them. For instance, using gzip or similar methods can significantly speed up the movement of large files.
* **Select the right integration runtime:** The choice of integration runtime (Azure IR, Self-hosted IR, or SSIS IR) is critical in optimizing performance. For example, self-hosted IR can be used for on-premises data movement to ensure secure and efficient transfers, while Azure IR is ideal for cloud-native operations.

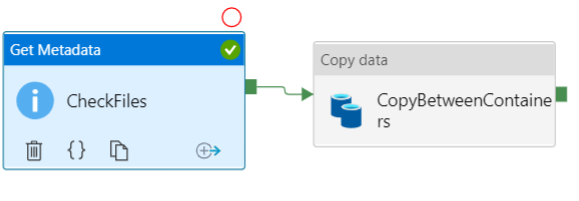
### **3. Implement robust error handling**

* **Retry policies:** Configure retry policies for transient errors, such as temporary network disruptions or server timeouts. This ensures pipelines can recover and complete successfully without manual intervention.
* **Set up alerts:** Implement alerts and notifications to proactively inform your team of pipeline failures or performance issues. Use tools like [**Azure Monitor**](https://learn.microsoft.com/en-us/azure/azure-monitor/overview) to configure custom alerts based on specific error types or execution delays, ensuring quick resolution and minimal downtime.

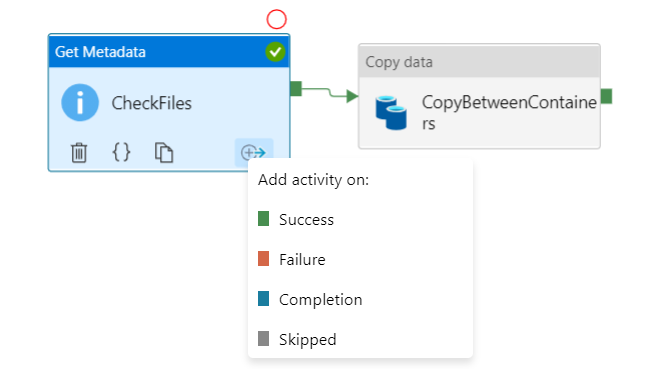
So, how is Azure Data Factory different from Databricks? If you are curious and want to discover the differences between Azure Data Factory and Databricks, check out [**Azure Data Factory vs Databricks: A Detailed Comparison blog**](https://www.datacamp.com/blog/azure-data-factory-vs-databricks).

Dependency between Activities

As mentioned previously, the default behavior of the pipeline activities is that it will not be executed unless the previous activity is executed successfully. This default dependency type in the Azure Data Factory activities is the **Success** type, with the green box icon and green arrow, as shown below:



If you click on any activity, you will see a number of icons that can be used to drop the activity, view the source code, and clone that activity. At the most right of these icons you will the  alone icon, which is used to control the dependency between the current activity and the next activities, where you can add and configure it to run the next activity if the current activity succeeded, failed, completed regardless of the result and finally skipped and not executed, as shown below:



To understand each activity execution dependency option from the previous list, let us create a more complex Azure Data Factory pipeline, in which we have a Get Metadata activity that checks the existence of a specific file in the source Azure Storage Account, if the file is in the storage account then the Get Metadata activity will be executed successfully, and the copy activity that is connected with the **green** box and the green arrow, that indicates a succeeded execution of the previous activity, will be executed too, and the file will be copied from the source Azure Storage Account to the destination account.

On the other hand, if the Get Metadata activity failed due to the absence of the file in the Azure Storage Account, the Lookup activity that is connected to the **red** box and the red arrow will be executed, to search for that file in the destination Azure Storage Account, and you can add any activity to perform the proper action if the file is found in the destination Azure Storage Account, based on your own logic.

You can see also a **blue** box and a blue arrow that is connected to a Stored Procedure execution activity, that contains the action that will be performed when the Get Metadata activity completed, regardless of the completion status, succeeded or failed, where we will use this activity to write a log to an Azure SQL Database table when the activity execution completed.

The last dependency option is the **grey** box and the grey arrow that is connected to the Stored Procedure execution activity that will be executed when the previous activity, which is the copy activity, is skipped and not executed, as shown below:

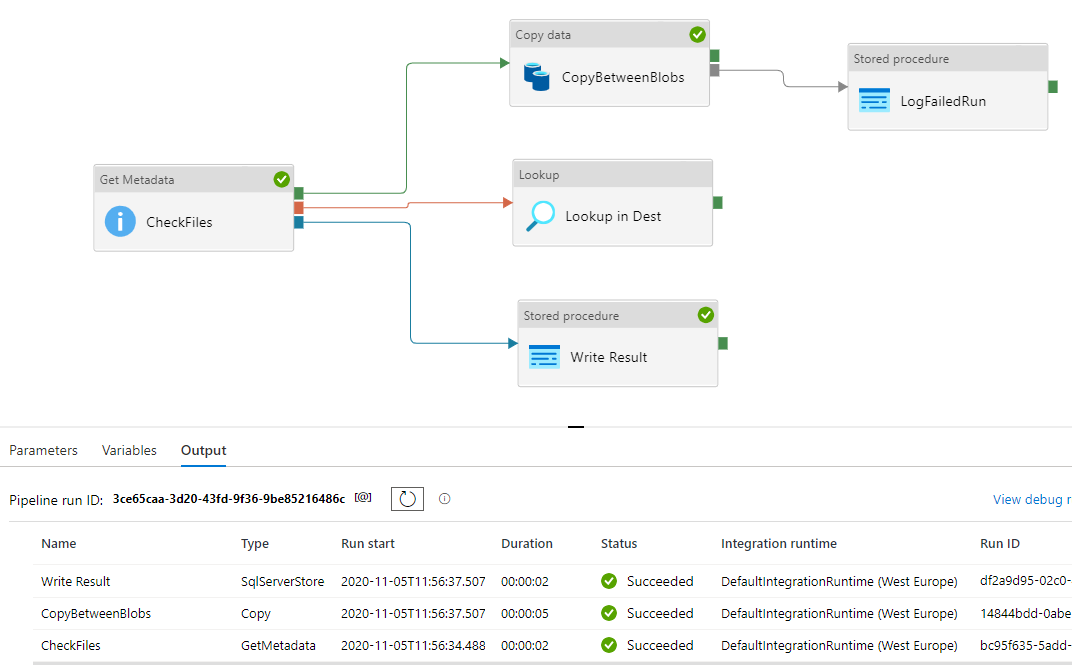


Now, let us test the Azure Data Factory pipeline execution scenarios and see what will be executed in each scenario!

Having the requested file in the source Azure Storage Account, we expect the Get Metadata activity to be executed successfully, which leads to executing the Copy activity that moves the file from the source storage account to the destination storage account.

Notice that the Lookup activity will be skipped as the Get Metadata succeeded and the Stored Procedure execution activity that is connected to the Copy activity also skipped as the Copy activity executed in this Azure Data Factory pipeline and not skipped.

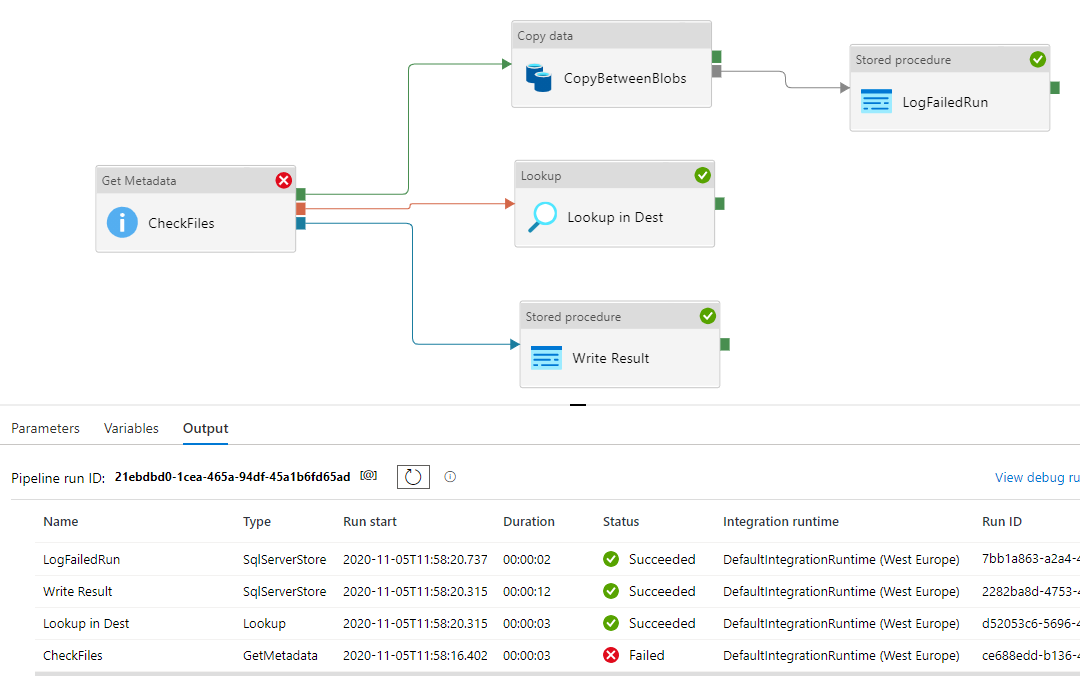
The Stored Procedure execution activity that is connected directly to the Get Metadata activity will be executed also, as the Get Metadata activity completed, as shown from the green circles and output tab below:



Now, assume that the requested file is not available in the source Azure Storage Account, then the Get Metadata activity will fail when executing the Azure Data Factory pipeline.

With the failure of the Get Metadata activity, the Lookup activity that searches for the file in the destination Azure Storage Account, and the Stored Procedure execution activity that is connected to the Copy activity will be executed also and log the Get Metadata activity failure in an Azure SQL Database table, as the Copy activity, that runs only when the Get Metadata activity completed successfully, is skipped due to the Get Metadata activity failure.

Again, the Stored Procedure execution activity that is connected directly to the Get Metadata activity will be executed also, as the Get Metadata activity completed, regardless of succeeded or failed, as shown from the green circles and output tab below:

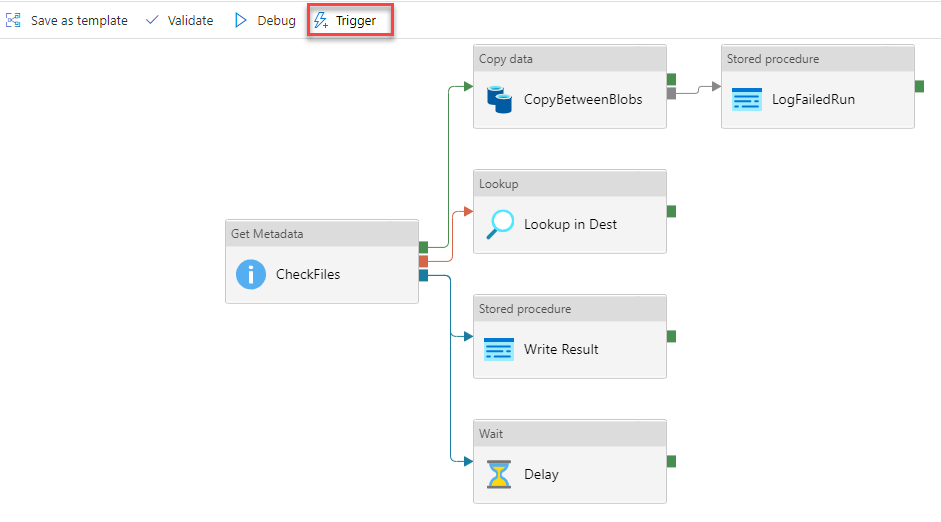


## Tumbling Window Trigger Dependency

In Azure Data Factory, Tumbling Window trigger consists of a series of fixed-size, non-overlapping, and contiguous time intervals that are fired at a periodic time interval from a specified start time, while retaining state. For more information, check How to Schedule Azure Data Factory Pipeline Execution Using Triggers.

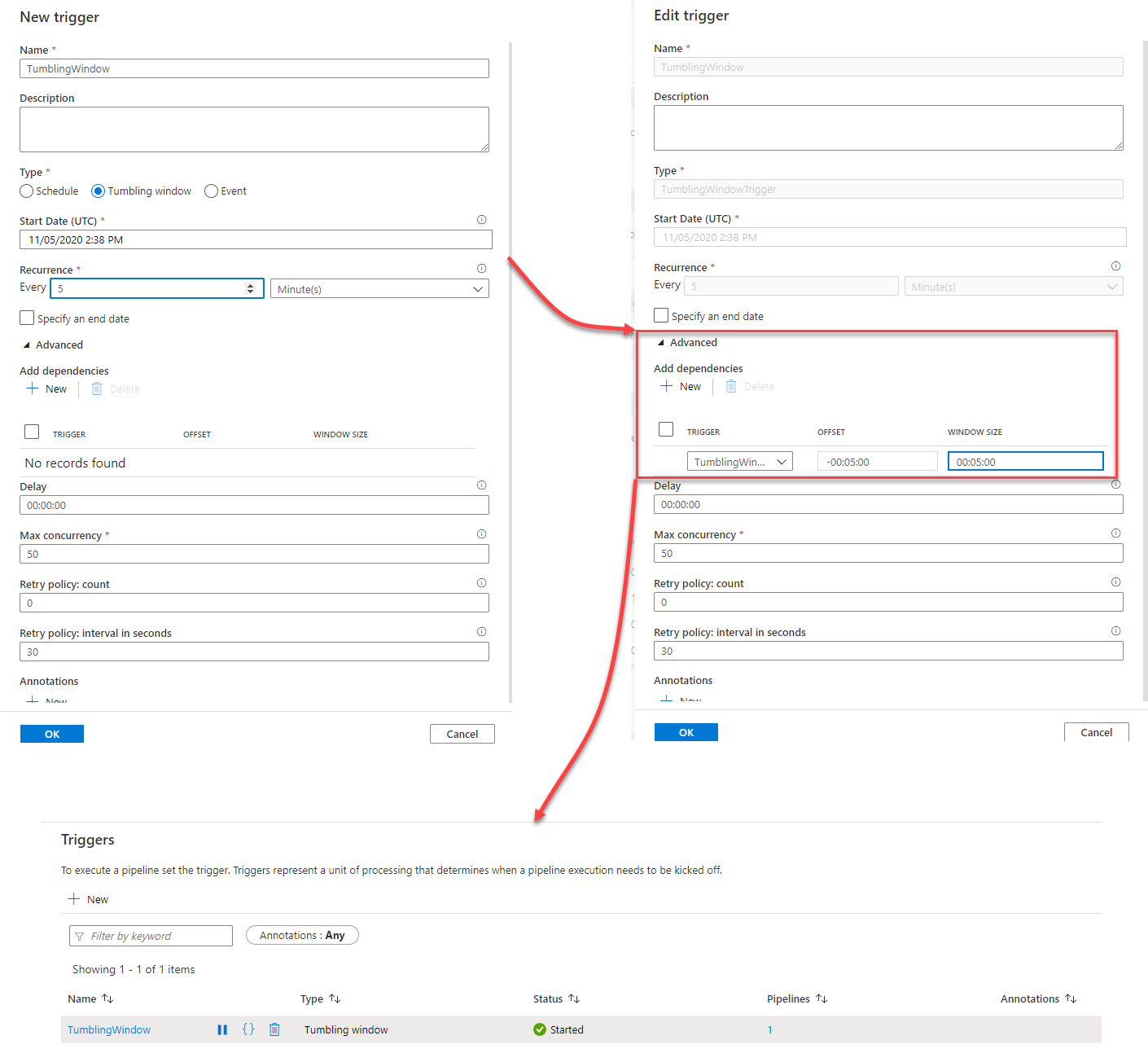
With Tumbling Window dependency, you can guarantee that the preceding window should be completed successfully in order to proceed with the next window, where the dependency in these windows can be with the preceding windows in the same trigger, called self-dependency, or dependency on another Tumbling Window trigger, based on the fact that, the Tumbling Window trigger in Azure Data Factory retains the execution status.

Let us first create a new Tumbling Window Trigger from the pipeline Author window, by clicking on the **Trigger** button and choose the **New/Edit** option, as shown below:

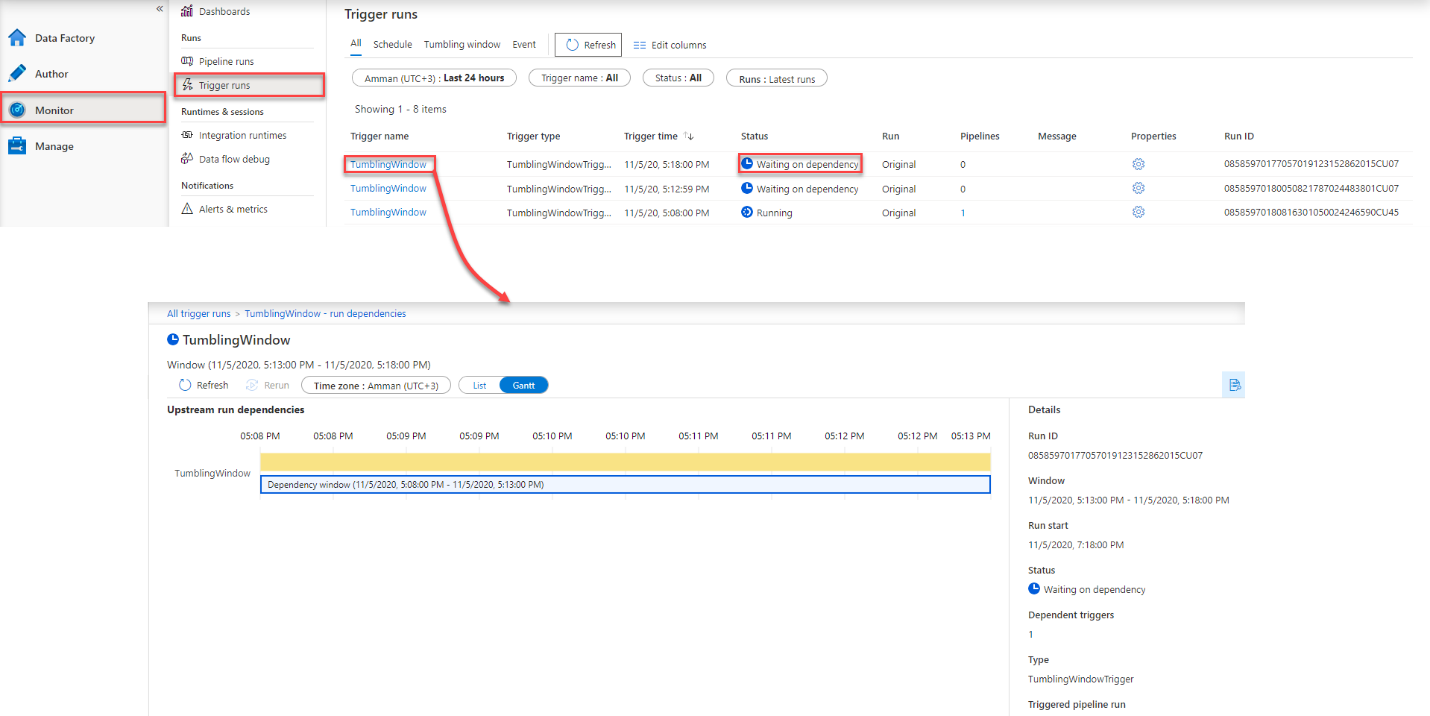


In the New Trigger window, provide a meaningful name for the trigger, choose Tumbling Window as the trigger type, set value for the pipeline start date, execution recurrence and optionally the end date, then click **Create** to proceed.

Once created, edit the trigger again, go to **Advanced** -> **Add Dependencies** then click on the **+New** option to add a new dependency. Here we will create a self-dependency to the same trigger so we will choose the same trigger in the trigger name. The **Offset** value is a positive or negative timespan value, which is mandatory for the self-dependency scenario that is used as an offset for the dependency trigger, where the **Size** is a positive timespan value that indicates the size of the dependency tumbling window. Both values should be provided in the hh:mm:ss format. After configuring the tumbling window trigger dependency, open the **Manage** page and browse for the Triggers list, where you can enable and edit the created trigger and check the Azure Data Factory pipelines that are connected to that trigger, as shown below:



To monitor the tumbling window execution, go to the **Monitor** page and check the status of the different pipeline executions in different tumbling windows, where you will see the first execution is in **Running** state and the rest of executions are in **Waiting** state on dependency, till the first time slot execution is completed successfully. You can see also that the name of the Tumbling Window trigger that is configured with dependencies is provided as a hyperlink that directs to the dependency monitoring window, that shows useful information about the trigger execution such as the Run ID, Window time, the execution status of the dependencies, taking into consideration that the tumbling window trigger will wait for 7 days before failing with timeout error, as shown below:



## **Conclusion**

Azure Data Factory simplifies the process of building, managing, and scaling data pipelines in the cloud. It provides an intuitive platform that caters to both technical and non-technical users, enabling them to integrate and transform data from various sources efficiently.

By leveraging its features, such as code-free pipeline authoring, integration capabilities, and monitoring tools, users can easily create scalable and reliable workflows.