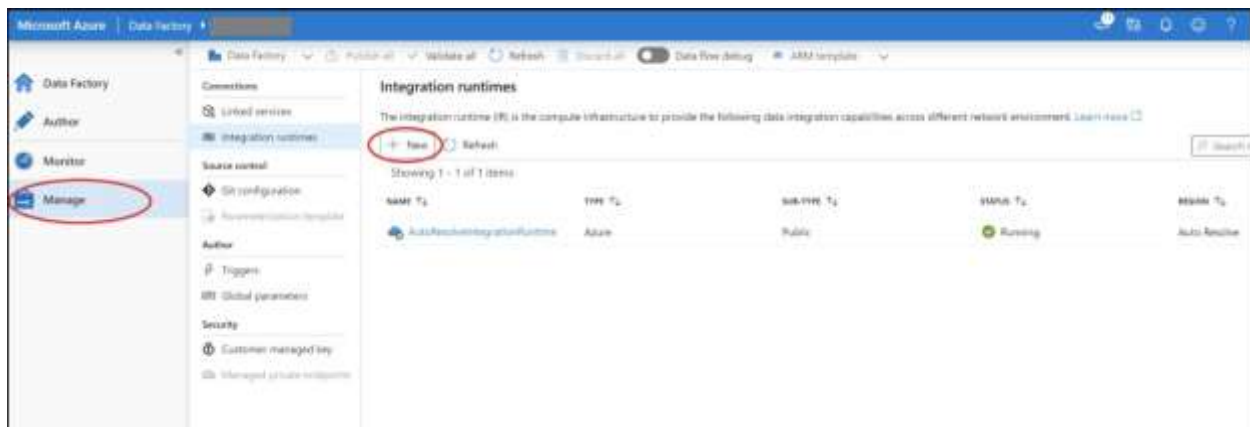



Create the Azure Integration Runtime



Integration runtime setup


Network environment:

Choose the network environment of the data source / destination or external compute to which the integration runtime will connect to for data flows, data movement or dispatch activities:



Azure

Use this for running data flows, data movement, external and pipeline activities in a fully managed, serverless compute in Azure.



Self-Hosted

Use this for running activities in an on-premise / private network

[View more](#) ▾

External Resources:

You can use an existing self-hosted integration runtime that exists in another resource. This way you can reuse your existing infrastructure where self-hosted integration runtime is setup.



Linked Self-Hosted

[Learn more](#) ⓘ

Integration runtime setup

The Data Factory manages the integration runtime in Azure to connect to required data source/destination or external compute in public network. The compute resource is elastic allocated based on performance requirement of activities.

Name *

Description

Type

Virtual network configuration (Preview)

☐ Disable ☒ Enable

☒ Enable interactive authoring capability after creation ⓘ

☒ Auto termination ⓘ

Terminate after minutes of inactivity

Region *

▸ Data flow run time

Add the source transformation

New linked service (Azure Data Lake Storage Gen2)

Name *

ADLSGen2

Description

Connect via integration runtime *

AutoResolveIntegrationRuntime (Managed Virtual Network)

[Edit integration runtime](#)

☒ Interactive authoring 

Authentication method

Account key

Account selection method

☒ From Azure subscription ☐ Enter manually

Azure subscription

<Select your Azure subscription>

Storage account name *

<Select your Storage account name>

Test connection

☒ To linked service ☐ To file path

Create

 Test connection

Cancel

Set properties

Name

MoviesDB

Linked service *

ADLSGen2

[Edit connection](#)



Interactive authoring 



File path

sample-data

/

Directory

/

moviesDB.csv

[Browse](#)



First row as header



Import schema



From connection/store



From sample file



None

▸ Advanced

OK

Back

Cancel

Add the filter transformation

The screenshot shows the Apache NiFi web console. At the top, a blue arrow-shaped node labeled 'MoviesDB' is shown with 'Columns: 6 total'. Below it is a dashed box labeled 'Add Source'. To the right, a search bar contains the text 'fil'. Below the search bar, a 'Row modifier' section is visible, containing a yellow funnel icon and the word 'Filter'. Below this, a tabbed interface shows 'Filter Settings' as the active tab. Under 'Filter Settings', there are three fields: 'Output stream name' with the value 'FilterYears', 'Incoming stream' with the value 'MoviesDB', and 'Filter on' with a large text area containing the placeholder 'Enter filter...'. A small 'ANY' button is visible in the top right corner of the text area.

Output stream name * [🔗](#) [📄](#)

Incoming stream * [MoviesDB](#)

Filter on *

Enter filter... ANY

Visual Expression Builder [Expression reference documentation](#)

FUNCTIONS

Filter

Functions: Input schema Parameters

movie
title
genres
year
Rating
Rotten Tomato

```
toInteger(year) >= 1910 && toInteger(year) <= 2000 && rlike(genres, 'Comedy')
```

Data preview Refresh

Output	year	genres
X	1980	Comedy
X	1902	Action/Adventure/fantasy/Sci-Fi
X	1915	Drama/War
X	1915	Drama
X	1915	Action/Adventure/Sci-Fi
✓	1917	Comedy
✓	1917	Comedy

Save and finish Cancel Clear contents


Use the following expression for the filter `toInteger(year) >= 1910 && toInteger(year) <= 2000 && rlike(genres, 'Comedy')` **Add the aggregate transformation**

The screenshot shows the Great Learning Data Studio interface. At the top, a pipeline is visualized with two components: 'MoviesDB' (Import data from MoviesDB) and 'FilterYears' (Columns: 6 total). Below this, a dashed box labeled 'Add Source' is visible. To the right, a 'Schema modifier' dropdown menu is open, showing 'Aggregate' as the selected option. Below the pipeline, the 'Aggregate Settings' tab is active. It displays 'Output stream name' as 'AggregateComedyRating' and 'Incoming stream' as 'FilterYears'. The 'Group by' tab is selected, showing 'FilterYears's column' as 'year' and 'Name as' as 'year'. Below this, the 'OUTPUT SCHEMA' section shows 'AverageComedyRating'. The 'FUNCTIONS' section shows a search for 'Filter...' with results for 'movie' and 'rating'. The 'EXPRESSION FOR FIELD "AVERAGECOMEDYRATING"' section shows the expression `avg(toInteger(Rating))`.

Use the following expression for this filter

avg(toInteger(Rating))

Apply the sink transformation



The diagram illustrates a data pipeline with three stages: **MoviesDB** (import data from MoviesDB), **FilterYears** (Filtering rows using expressions on columns 'year', 'genres'), and **AggregateComedyRating** (Columns: 3 total). A **Sink** transformation is being applied to the output of the **AggregateComedyRating** stage. The **Sink** transformation settings are shown below the diagram.

Sink Settings:

- Output stream name ***: Sink
- Incoming stream ***: AggregateComedyRating
- Sink dataset ***: Select... (dropdown menu)
- Options**:
 - ☒ Allow schema drift
 - ☐ Validate schema

[Documentation](#)

[+ New](#)

Set properties

Name

MoviesSink

Linked service *

ADLSGen2

[Edit connection](#)



Interactive authoring 



File path

sample-data

/

output

/

File

[Browse](#)



First row as header



Import schema



From connection/store



From sample file



None

▶ Advanced

OK

Back

Cancel